INTRODUCTION – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

Ollscoil na Gaillimhe
University of Galway

Module overview (subject to change)

- **Thursday IT 101 1-3 PM**

- Software Engineering (**Dr. Enda Barrett**) – S1 + S2
- Group project – (**Dr. Enda Barrett**) – S1 + S2

- Blackboard
  - Notes
  - Problem sheets
  - Assignment submission
  - Announcements

- Lab Tutors: Daniel Kelly
  - Labs start in a couple of weeks – **Friday 12 - 2PM IT 106**

Module Details

- Exam at the end of the year (Summer 2023)
  - No exam at the end of Semester 1
  - 4 questions answer 3

- Group project will account for **40%** of the final mark
  - Delivery of a spec – 5% - Group
  - Project Demo and Final Report (last week of term)
  - Graded holistically at the end

Semester 1 goal

- Learn about the software engineering principles and methods which enable the building of large team based software systems
- Understand the importance of version control and team based development
- Get cloud services experience and configure the deployment environment for your group based projects

Blackstone Launchpad

- Blackstone LaunchPad

Blackstone LaunchPad

Group Projects

- Groups of 4 people
  - Web based application i.e. You will build a **web application** using (HTML, CSS, JavaScript), you will build the backend in **Node.js**, deploy to Firebase and utilise a data storage component i.e. **Firestore**
  - Past projects included
    - Chat rooms
    - Personal Dashboards
    - Photo sharing application

  - Some difference between groups!
  - So get thinking about what you would like to do!
  - I wish to keep the project groups within the class splits during the face to face slots where possible.

Group project dates

- Please form your project groupings by

  Friday 23rd September at 17:00

- Nominated group lead should email me (Enda.Barrett@nuigalway.ie) the following
  - Team/Group name ("The coders"),
  - Names of each member,
  - Group sizes of 4 work best
  - Come up with an idea and mail it to me
    - real time event app
    - instant messenger
    - social media tool
  - If you don't have a group I will randomly assign you
- If you opt out you will get 0!

Web applications

- Clear separation of concerns
  - Frontend view code or UI (CSS, HTML)
    - Look and feel, structuring content
  - Frontend dynamic content (JavaScript, VueJS (potentially))
    - GET/POST methods, handling/updating data
  - Backend server side code (Node.js) – **Firebase functions**
    - Returning data, developing APIs
  - DB component (**Firestore**)
    - Schemas, queries for document retrieval
- Basic app up and running by Christmas deployed using Firebase.

CT216 SOFTWARE ENGINEERING 1

CLOUD COMPUTING

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

Ollscoil na Gaillimhe
University of Galway

What is cloud computing?

- "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (NIST, May 2011)

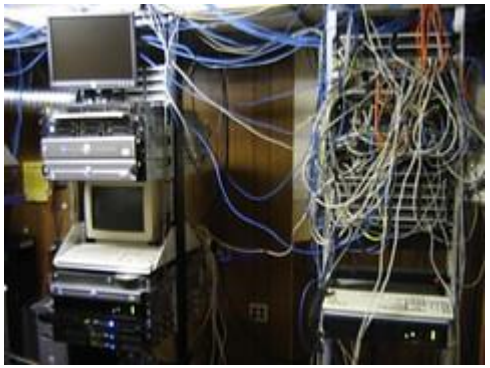
What is cloud computing

- Back in the early 00's there was this bookseller called Amazon, who made their money by shipping books (and everything else) around the world.
- To support their web app (Amazon.com) they had built up some neat hosting infrastructure and software to manage it at scale with a couple of data centers
- The centers were somewhat underutilised and Amazon decided to start selling this spare capacity.

Server room

- At that time most businesses/organisations maintained a server room on premises. In it they would have separate rack mounted PCs/servers.
  - Data would be stored on large Storage Networks
  - Backups would be run on this data
  - Multiple machines (servers) would run business critical software

- There were challenges to this
  - Maintenance
  - Upgrading machines
  - Upfront purchasing costs
  - Hire staff to manage it (Sysadmins)



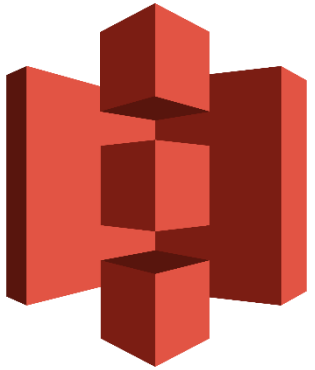
What is cloud computing

- Started with storage
  - Simple Storage Service launched on **March 14 2006** marking the beginning of Amazon Web Services
  - This allowed users to store documents, files, data on an S3 bucket without having to manage, purchase, maintain the underlying disk hardware.

What is cloud computing

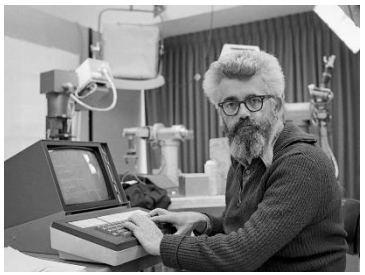
- Then came computing
  - It followed up its successful storage launch of S3 with EC2 or Elastic Compute Cloud in August 2006
  - This allowed you to have access to a remote server accessible via the internet!

Demand was strong…



History of cloud computing

- Computing may someday be organized as a public utility just as the telephone system is a public utility," Professor John McCarthy said at MIT's centennial celebration in 1961. "Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system … Certain subscribers might offer service to other subscribers … The computer utility could become the basis of a new and important industry."



https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/

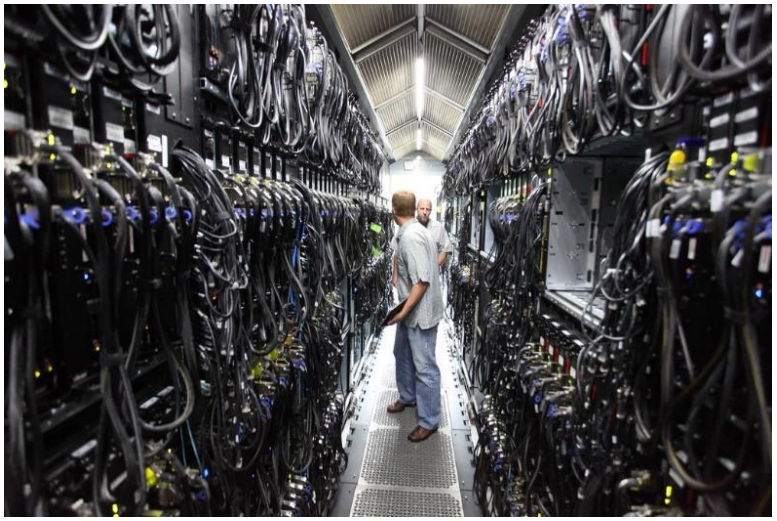
Cloud computing growth

- The growth in cloud computing over the past decade has been phenomenal.
- In April 2011, Forrester projected that it would be worth $160bn dollars by 2020, in reality it was 27% larger at $219bn.
- In 2022 it hit over $480bn in value and is projected to exceed $1tn by 2029.

https://www.fortunebusinessinsights.com/cloud-computing-market-102697

Where is the cloud?

Amazon US-East N. Virginia



Cloud types – Public cloud

- Amazon, MS Azure, Google Cloud are examples of public clouds.
- Any member of the public can sign up and start provisioning compute resources within minutes
- They are highly scalable and allow an organisation to grow its infrastructure rapidly

Cloud types – Private cloud

- Private cloud
  - Computing resources are dedicated to a single customer and not shared with other customers. Considered to be more secure.
  - AWS do offer a virtual private cloud
  - https://aws.amazon.com/vpc/
  - Organisations can also host their on cloud on-prem using software such as OpenStack.

Cloud types - Hybrid

- Finally a hybrid cloud is simply a mix of public cloud resources and private resources. An organisation may choose this option if there is a mixture in the criticality of their data or computational requirements.
- Data that doesn't require heightened security can be pushed on to the public cloud whereas that which does can be hosted on the private cloud.

Cloud services

- **Software as a Service** (SaaS) - provides users with—essentially—a cloud application, the platform on which it runs, and the platform's underlying infrastructure.
- **Platform as a Service** (PaaS) - provides users with a platform on which applications can run, as well as all the IT infrastructure required for it to run.
- **Infrastructure as a Service** (IaaS) - provides users with compute, networking, and storage resources.

Virtual Servers

- Infrastructure as a Service (IaaS)
  - Amazon, Google, Microsoft
- Create virtual machines
  - t1.micro, m1.small, c1.medium, m1.large…
- Customise instances and add greater amounts of storage.
- Each instance can be booted up with a different AMI, you can even create your own!
- Xen Hypervisor (Sun, AMD, IBM, Dell, Intel)
- Storage area networks provide the storage

Advantages/Disadvantages of cloud computing

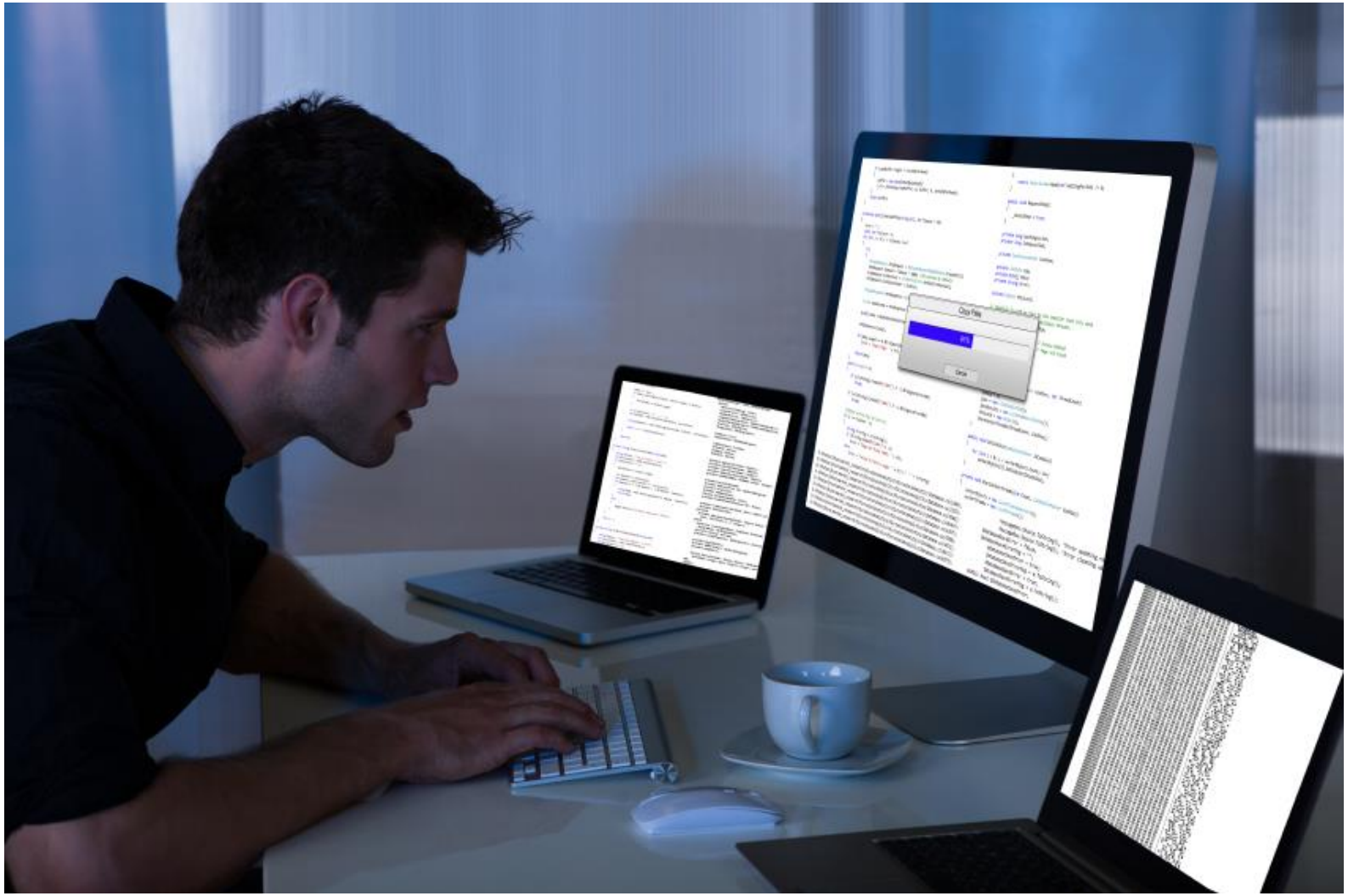
- When compared to hosting in-house cloud computing has a number of benefits
  - Elasticity – if your application becomes very popular you can procure new resources in minutes
  - Reduced capital expenditure
  - Economies of scale
- There are also some drawbacks
  - Security/privacy
  - Cost
  - Migration issues

SOFTWARE PROCESSES

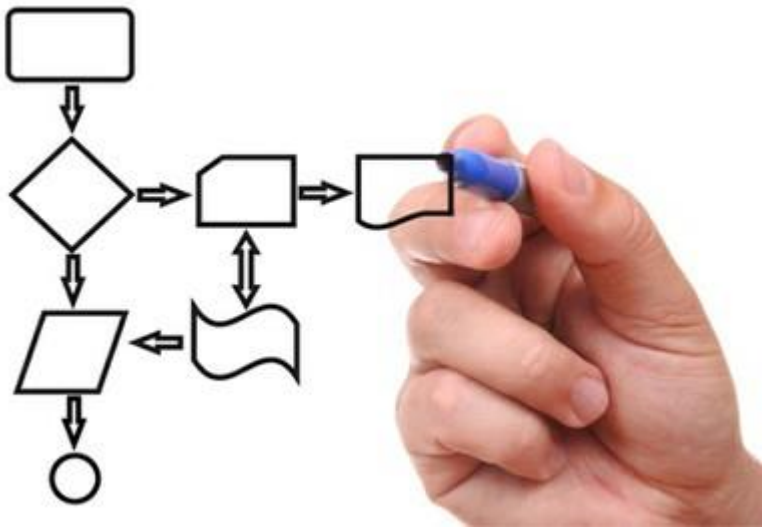
Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

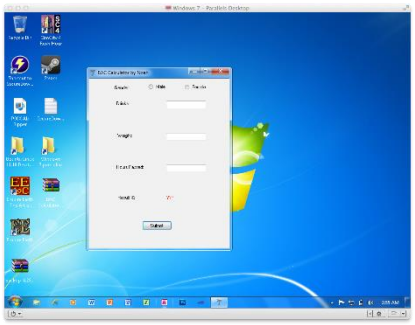
A Software Process: Who is like this?



Is there anyone like this?

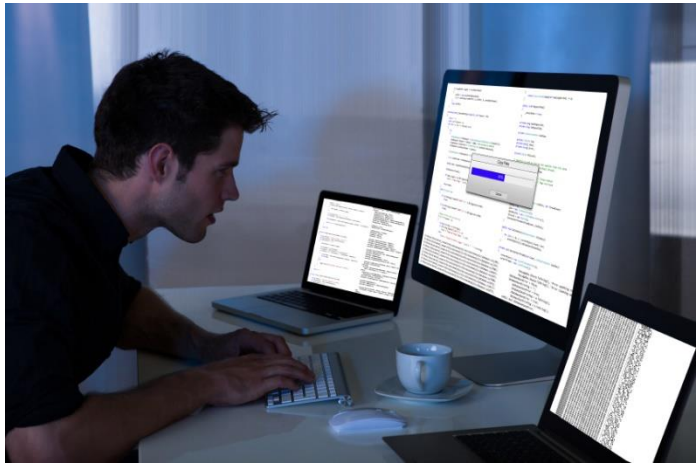


Recap: Software Dev. is complex and varied

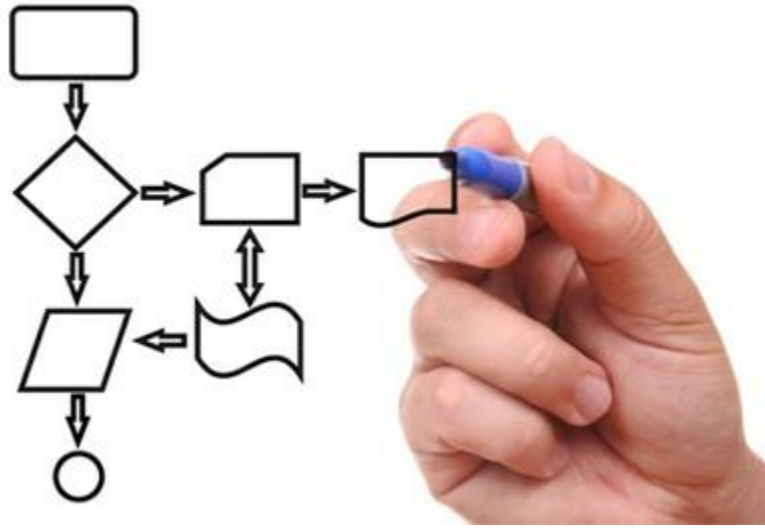


- Ada
- 3 levels of redundancy
- Different dev teams

Difference between these two?



vs



Bad Process

Bad Engineer

Good Process

Good Engineer

Building a house

- Plan
  - Sketch the layout/structure
  - Determine how the components will fit

- Construction
  - Laying foundations/block laying/engineer testing

- Deployment
  - Delivered to the customer who provides feedbac
    list

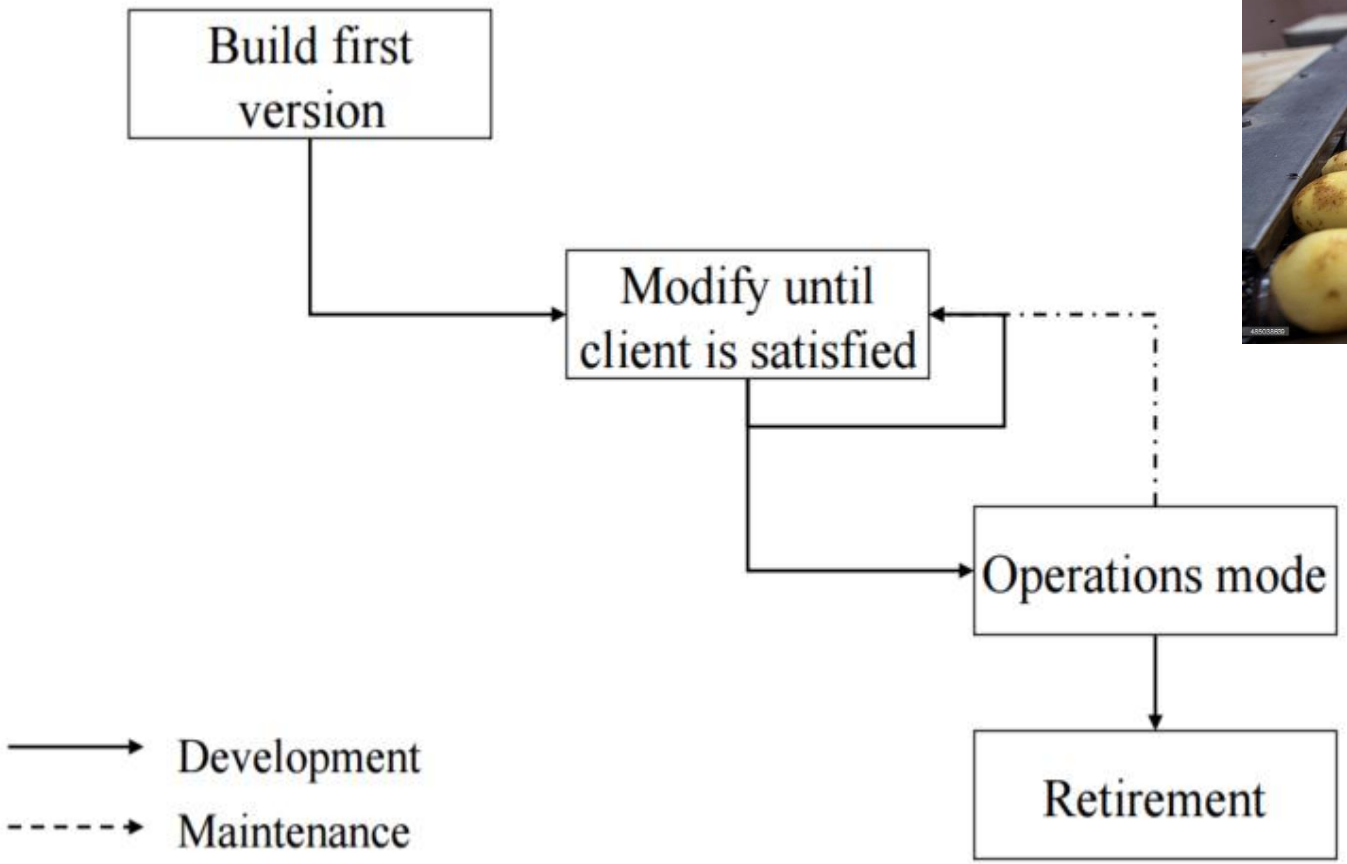
The software process

- A structured set of activities required to develop a software system
- Four fundamental process activities
  - Specification
  - Development
  - Validation
  - Evolution
- The foundation of software engineering is in the **process**
- **Goal:** To efficiently and predictably deliver a product that meets the requirements

Motivating case

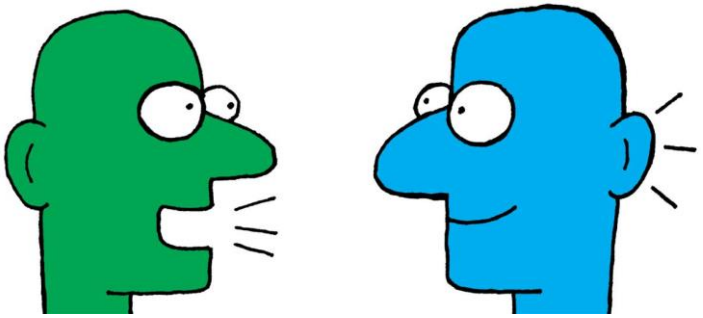
- You've been hired by a local independent retailer to build their potato peeling system

Build and fix model…worst approach



Software Process Model

- 1) Software Specification
  - Talk to the customer
  - Understand the problems
  - Talk to any relevant stakeholders



- 2) Software Development
  - Map out the tasks
  - Design the software
  - Develop the solution

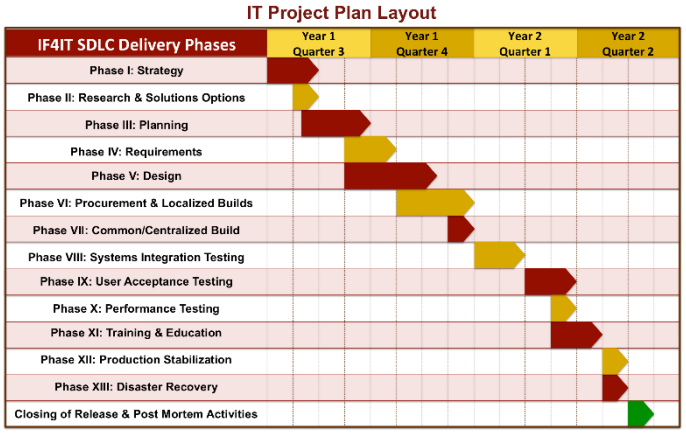


Figure: Example Layout of SDLC Phases as Key IT Project Milestones

Software Process Model…

- 3) Software validation
  - Does it meet requirements
  - Is it what the customer wanted

- 4) Software evolution (maintenance)
  - Modified to adapt
  - Changes in requirements
  - Customer & Market conditions

Software Engineering Practice

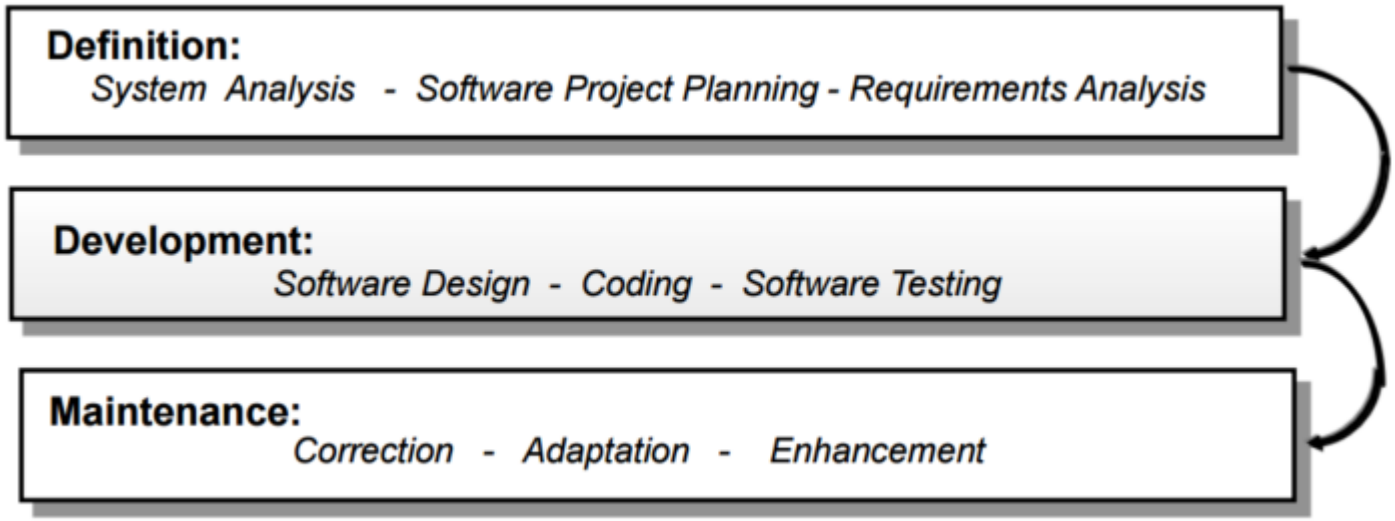
- 1) **Understand the problem** (*Communication and analysis*)
  - Who are the stakeholders?
  - What are the unknowns?
- 2) **Plan the solution** (*Modelling and software design*)
  - Have we seen this problem before?
  - Has a similar problem been already solved? Plagiarism
  - Can sub problems be found?
- 3) **Carry out the plan** (*Write the code*)
  - Does the solution conform to the plan?
  - Has the code been reviewed for correctness?
- 4) **Examine the result** (*Test it*)
  - Is each component testable?
  - Does the solution produce results as defined originally?

General Software Engineering Questions

- **What** is the problem to be solved?
  - Requirements definition
- **What** are the characteristics of the software (system) used to solve the problem?
  - Analysis
- **How** will the system be realised/constructed?
  - Design
- **How** will design and construction errors be uncovered and dealt with?
  - Test
- **How** will the system be supported long-term?
  - Maintenance

Overview of Software Engineering

- There are three generic phases, regardless of paradigm:
  - Definition, **a focus on the *What*.**
  - Development, **a focus on the *How*.**
  - *Maintenance,* ***focuses on Change***



Software Engineering should…

- Provide a clear statement of the project mandate & objectives;
- Create effective means of communication;
- Increase user involvement & ownership;
- Provide an effective management framework to support productivity & pragmatism;
- Establish quality assurance procedures;
- Provide sound resource estimation and allocation procedures;
- Ensure the effectiveness and durability of systems produced;
- Encourage the re-usability of code and/or solutions;
- Reduce the organisation's vulnerability to the loss of software development personnel (MJ);
- Reduce and support post implementation maintenance of systems;

CT216 SOFTWARE ENGINEERING 1

CLOUD COMPUTING

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

Ollscoil na Gaillimhe
University of Galway

What is cloud computing?

- "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." (NIST, May 2011)

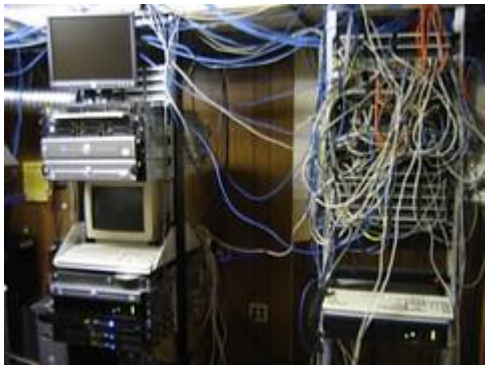
What is cloud computing

- Back in the early 00's there was this bookseller called Amazon, who made their money by shipping books (and everything else) around the world.
- To support their web app (Amazon.com) they had built up some neat hosting infrastructure and software to manage it at scale with a couple of data centers
- The centers were somewhat underutilised and Amazon decided to start selling this spare capacity.

Server room

- At that time most businesses/organisations maintained a server room on premises. In it they would have separate rack mounted PCs/servers.
  - Data would be stored on large Storage Networks
  - Backups would be run on this data
  - Multiple machines (servers) would run business critical software

- There were challenges to this
  - Maintenance
  - Upgrading machines
  - Upfront purchasing costs
  - Hire staff to manage it (Sysadmins)



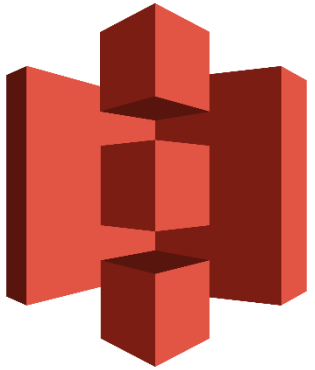
What is cloud computing

- Started with storage
  - Simple Storage Service launched on **March 14 2006** marking the beginning of Amazon Web Services
  - This allowed users to store documents, files, data on an S3 bucket without having to manage, purchase, maintain the underlying disk hardware.

What is cloud computing

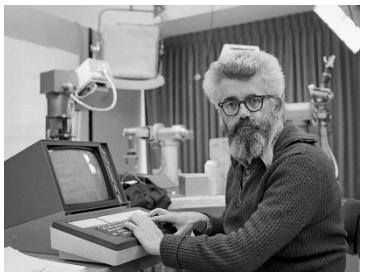
- Then came computing
  - It followed up its successful storage launch of S3 with EC2 or Elastic Compute Cloud in August 2006
  - This allowed you to have access to a remote server accessible via the internet!

Demand was strong…



History of cloud computing

- Computing may someday be organized as a public utility just as the telephone system is a public utility," Professor John McCarthy said at MIT's centennial celebration in 1961. "Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system … Certain subscribers might offer service to other subscribers … The computer utility could become the basis of a new and important industry."



https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/

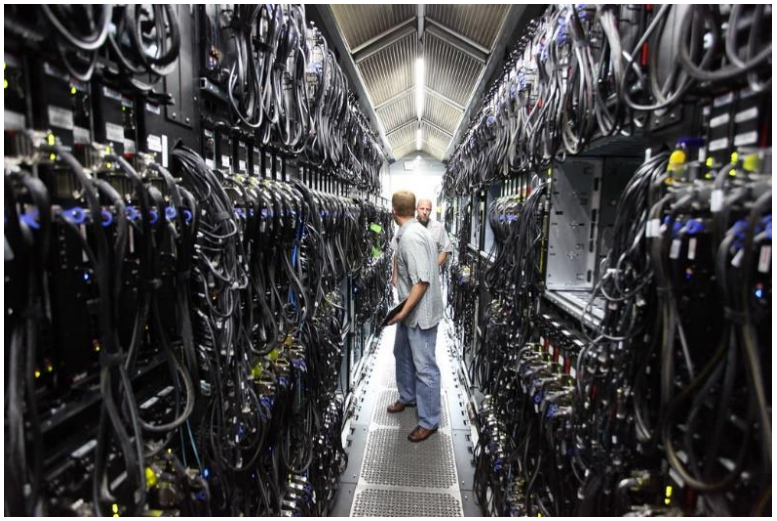
Cloud computing growth

- The growth in cloud computing over the past decade has been phenomenal.
- In April 2011, Forrester projected that it would be worth $160bn dollars by 2020, in reality it was 27% larger at $219bn.
- In 2022 it hit over $480bn in value and is projected to exceed $1tn by 2029.

https://www.fortunebusinessinsights.com/cloud-computing-market-102697

Where is the cloud?

Amazon US-East N. Virginia



Cloud types – Public cloud

- Amazon, MS Azure, Google Cloud are examples of public clouds.
- Any member of the public can sign up and start provisioning compute resources within minutes
- They are highly scalable and allow an organisation to grow its infrastructure rapidly

Cloud types – Private cloud

- Private cloud
  - Computing resources are dedicated to a single customer and not shared with other customers. Considered to be more secure.
  - AWS do offer a virtual private cloud
  - https://aws.amazon.com/vpc/
  - Organisations can also host their on cloud on-prem using software such as OpenStack.

Cloud types - Hybrid

- Finally a hybrid cloud is simply a mix of public cloud resources and private resources. An organisation may choose this option if there is a mixture in the criticality of their data or computational requirements.
- Data that doesn't require heightened security can be pushed on to the public cloud whereas that which does can be hosted on the private cloud.

Cloud services

- **Software as a Service** (SaaS) - provides users with—essentially—a cloud application, the platform on which it runs, and the platform's underlying infrastructure.
- **Platform as a Service** (PaaS) - provides users with a platform on which applications can run, as well as all the IT infrastructure required for it to run.
- **Infrastructure as a Service** (IaaS) - provides users with compute, networking, and storage resources.

Virtual Servers

- Infrastructure as a Service (IaaS)
  - Amazon, Google, Microsoft
- Create virtual machines
  - t1.micro, m1.small, c1.medium, m1.large…
- Customise instances and add greater amounts of storage.
- Each instance can be booted up with a different AMI, you can even create your own!
- Xen Hypervisor (Sun, AMD, IBM, Dell, Intel)
- Storage area networks provide the storage

Advantages/Disadvantages of cloud computing

- When compared to hosting in-house cloud computing has a number of benefits
  - Elasticity – if your application becomes very popular you can procure new resources in minutes
  - Reduced capital expenditure
  - Economies of scale
- There are also some drawbacks
  - Security/privacy
  - Cost
  - Migration issues

SCRUM – ROLES AND CEREMONIES

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

Scrum Framework

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Product Owner

- Define the features of the product
  - Tries to remove conjecture, “I know the customer wants this” as opposed to “I believe this would be a good feature”
- Decide on release date and content
  - Usually responsible for press releases
- Be responsible for the profitability of the product (ROI)
- Prioritize features according to market value
  - Conduct market research, feasibility studies
- Adjust features and priority every iteration, as needed
- Accept or reject work results.

Scrum Master

- Represents management to the project
  - Often one of the engineers
- Responsible for enacting Scrum values and practices
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

Scrum Team

- Typically 5-10 people
- Cross-functional
- QA, Programmers, UI Designers, etc.
- Members should be full-time
- May be exceptions (e.g., System Admin, etc.)
- Teams are self-organizing
- Membership can change only between sprints

Roles

- Product owner
- ScrumMaster
- Team

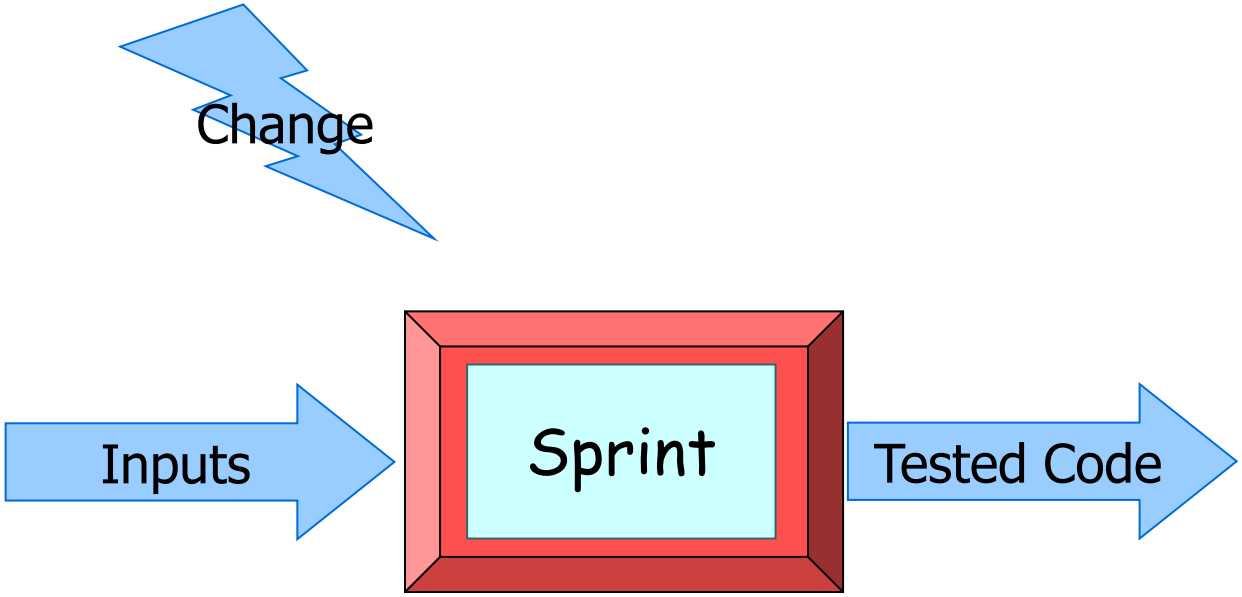
Ceremonies

- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Sprints

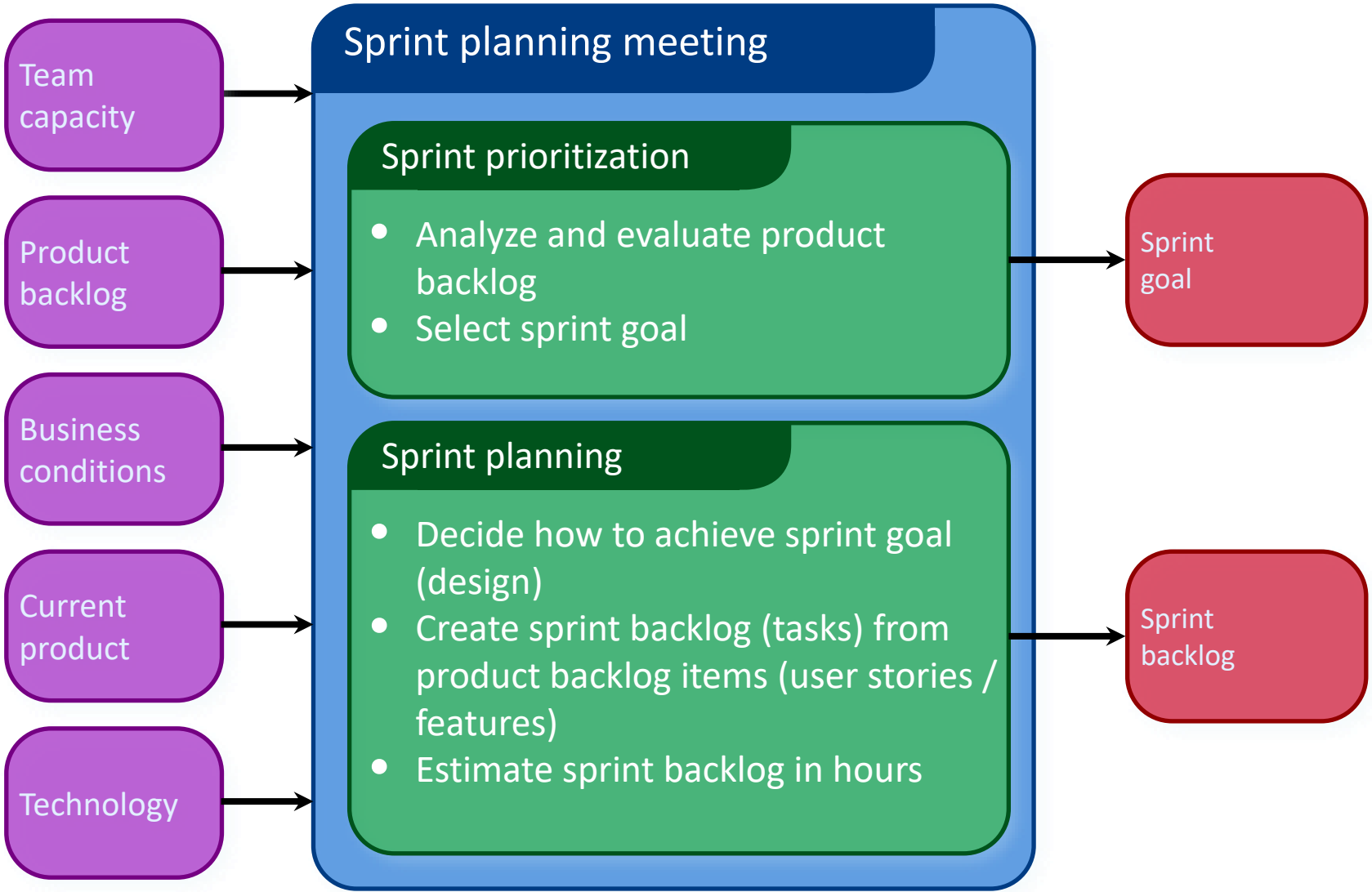
- Scrum projects make progress in a series of "sprints"
- Target duration is one month
  - +/- a week or two (2 - 6 weeks max)
    - But, a constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint
  - The output is a build which may or may not be a release
- Move onto the next sprint…

No changes during sprint



- Plan sprint durations around how long you can commit to keeping change out of the sprint

Sprint planning



Sprint planning

- Team selects items from the product backlog that they can commit to completing
- Sprint backlog is created
  - Tasks are identified and each is estimated (1-16 hrs)
  - Team collaboratively does this

As a vacation planner, I want to see photos of the hotels.

Code the middle tier (8 hours)
Code the user interface (4)
Write test fixtures (4)
Code the foo class (6)
Update performance tests (4)

What is the Product Backlog

- A list of all desired work on the project (the requirements)
  - Usually a combination of
    - story-based work ("let user search and replace")
    - task-based work ("improve exception handling")
- List is prioritized by the Product Owner
  - Typically a Product Manager, Marketing, Internal Customer, etc.
  - Priority groupings (high, medium, low … etc)
  - Reprioritised at the start of each sprint
  - Spreadsheet (usually)

To create a Sprint Backlog you must have a (Sprint) goal

Life Sciences

Support features necessary for population genetics studies.

Database Application

Make the application run on SQL Server in addition to Oracle.

Financial services

Support more technical indicators than company ABC with real-time, streaming data.

From Sprint Goal to Sprint Backlog

- Scrum team takes the Sprint Goal and decides what tasks are necessary
- Team self-organizes around how they will meet the Sprint Goal
  - Manager does not assign tasks to individuals
- Managers don't make decisions for the team

- Sprint Backlog is created

Sprint backlogs during the sprint

- Changes
  - Team adds new tasks whenever they need to, in order to meet the Sprint Goal
  - Team can remove unnecessary tasks
  - But: Sprint Backlog can only be updated by the team

- Estimates are updated whenever there's new information

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

- ~~Sprints~~
- ~~Sprint planning~~
- Sprint review
- Sprint retrospective
- ~~Daily scrum meeting~~

Sprint review meeting

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - Two hour prep time
  - No slides
- Participants
  - Customers
  - Management
  - Product Owners
  - Engineering team

Sprint Retrospective meetings

- Typically 15–30 minutes
- Done after every sprint
- Feedback meeting – time to reflect on how things are going…

- Many participants
  - ScrumMaster
  - Product owner
  - Team
  - Possibly customers and others

Start/Stop/Continue

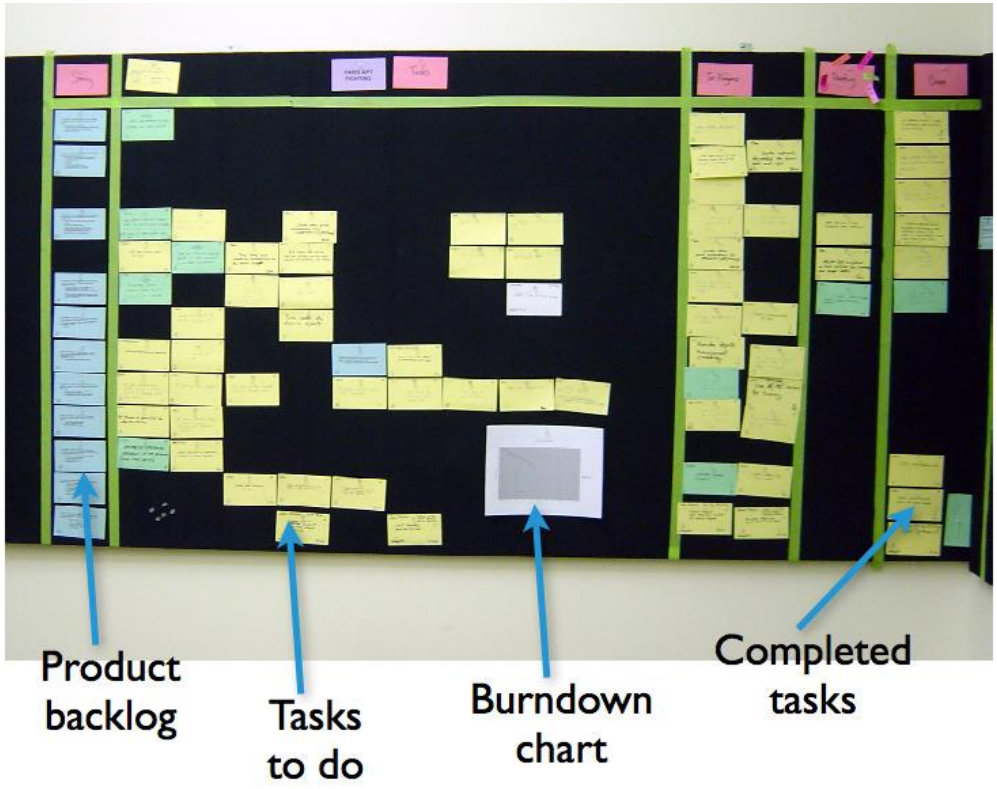
Whole team gathers and discusses what they'd like to:

Start doing

Stop doing

Continue doing

War room


Product backlog
Tasks to do
Burndown chart
Completed tasks

Pros/Cons of Agile Methods

- *Advantages*
  - *Completely developed and tested features in short iterations*
  - *Simplicity of the process*
  - *Clearly defined rules*
  - *Increasing productivity*
  - *Self-organizing*
  - *Each team member carries a lot of responsibility*
  - *Improved communication*
  - *Combination with Extreme Programming*
- *Drawbacks*
  - *"Undisciplined hacking" (no written documentation)*
  - *Violation of responsibility*
  - *Current mainly carried by the inventors*
  - *Employee Burnout & Fatigue.*

SOFTWARE DEVELOPMENT PARADIGMS –AGILE METHODS

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

Agile software development

- What is agile software development?
  - Scrum – Software Project Management Methodology
  - XP – Software Development Methodology

Software Development Lifecycle

- The software lifecycle is an abstract representation of a software process. It defines the steps, methods, tools, activities and deliverables of a software development project. The following **lifecycle phases** are considered:
  - 1. requirements analysis
  - 2. system design
  - 3. implementation
  - 4. integration and deployment
  - 5. operation and maintenance

SDLC Limitations

- Classical project planning methods have a lot of disadvantages
  - Huge efforts during the planning phase {Requirements + Design}
  - Poor requirements conversion in a rapidly changing environment
  - Treatment of staff as a factor of production

Agile Man

Agile Motivations

- Agile proponents argue:
  - Software development processes relying on lifecycle models are too heavyweight or cumbersome
  - Too many things are done that are not directly related to the software product being produced, i.e. design, models, requirements docs, documentation that isn't shipped as part of the product
  - Difficulty with incomplete or changing requirements
  - Short development cycles (Mobile Apps)
  - More active customer involvement needed

What is Agile?

- Agile methods focus on
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- Several agile methods
  - No single agile method
  - Scrum
  - XP
- No single definition
- Agile Manifesto closest to a definition
  - Set of principles
  - Developed by Agile Alliance (http://www.agilealliance.org/)

Agile methods

- Agile methods:
  - Scrum
  - Extreme Programming (XP)
    - Continuous Integration
    - Test Driven Development (TDD)
    - …
- Agile Alliance (www.**agilealliance**.org)
  - A non-profit organization promotes agile development

Scrum in 100 words

- Scrum is an agile project management methodology for managing product development.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. The teams self-manage to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance for another iteration.

History of Scrum

- 1995:
  - Analysis of common software development processes found that they are not suitable for unpredictable and non-repeatable processes
  - Design of a new method: Scrum by Jeff Sutherland & Ken Schwaber
  - Enhancement of Scrum by Mike Beedle & combination of Scrum with Extreme Programming
- 1996:
  - Introduction of Scrum at the (Object-Oriented Programming, Systems, Languages & Applications) OOPSLA conference
- 2001:
  - Publication “Agile Software Development with Scrum” by Ken Schwaber & Mike Beedle
  - Gained in popularity steadily ever since
- Founders are members in the Agile Alliance

Characteristics of Scrum

- Self-organizing teams
  - No need for project manager (in-theory)
- Product progresses in a series of month-long "sprints"…could be biweekly also
- Assumes that the software cannot be well defined and requirements will change frequently
- Requirements are captured as items in a list of "product backlog"
- No specific engineering practices prescribed
  - XP, TDD, FDD…
- Best approach is to start with Scrum and then invent your own version using XP, TDD, FDD

Daily Scrum/Standup

- Parameters
  - Daily
  - 15-minutes
  - Stand-up
  - Not for problem solving
  - Only team members, ScrumMaster, Product Owners should talk
  - Should help to avoid additional unnecessary meetings
  - Commitment in front of peers to complete tasks

Answer three questions

1. What did you do yesterday?
2. What will you do today?
3. Is anything in your way?

Daily SCRUM/Standup

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the team

AGILE METHODS – EXTREME PROGRAMMING

Dr. Enda Barrett

OÉ Gaillimh
NUI Galway

Overview

- **XP**
  - **General concepts**
- Specific XP concepts
  - Pair programming
  - Test Driven Development
  - Continuous Integration

Scrum Summary

- Scrum is a **project management** methodology

Characteristics of Scrum

- Self-organizing teams
  - No need for project manager (in-theory)
- Product progresses in a series of month-long "sprints"…could be biweekly also
- Assumes that the software cannot be well defined and requirements will change frequently
- Requirements are captured as items in a list of "product backlog"
- No specific engineering practices prescribed
  - XP, TDD, FDD…
- Best approach is to start with Scrum and then invent your own version using XP, TDD, FDD

Daily Scrum/Standup

- Parameters
  - Daily
  - 15-minutes
  - Stand-up
  - Not for problem solving
  - Only team members, ScrumMaster, Product Owners should talk
  - Should help to avoid additional unnecessary meetings
  - Commitment in front of peers to complete tasks

Answer three questions

1. What did you do yesterday?
2. What will you do today?
3. Is anything in your way?

Daily SCRUM/Standup

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the team

Scrum Framework

Roles

- Product owner
- ScrumMaster
- Team

Ceremonies

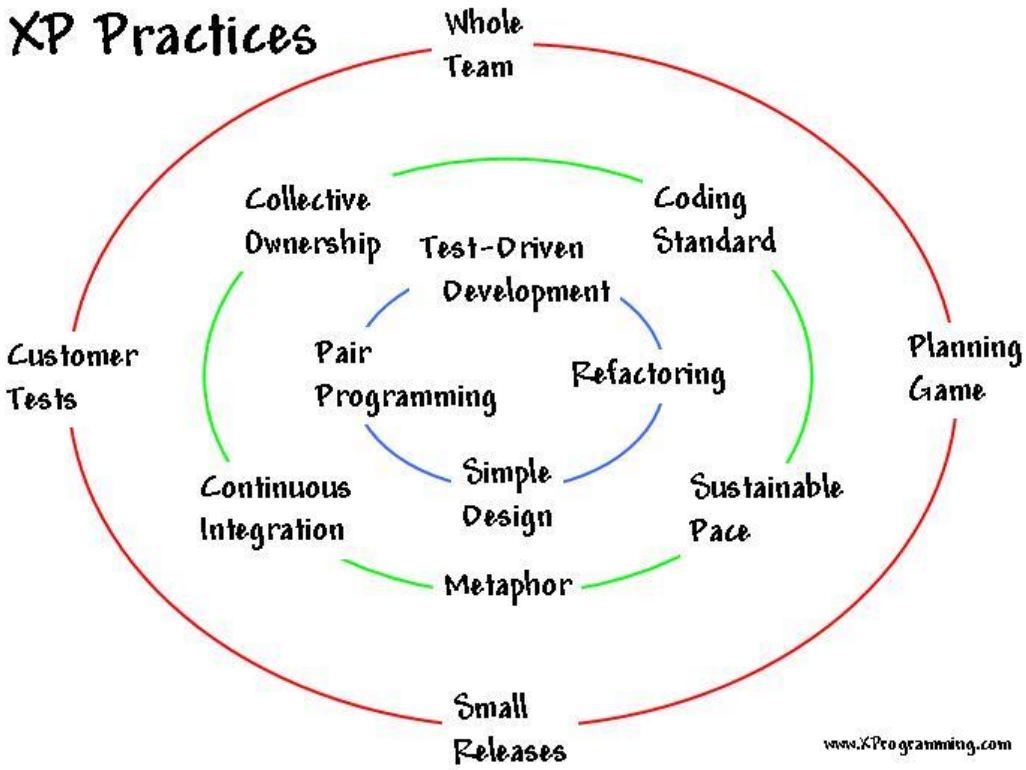
- Sprints
- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

We need an Agile Development method

- eXtreme Programming (XP)
  - One of the most popular agile **software development** methods

eXtreme Programming

- Pair programming
- Refactoring
- Test Driven Development
- Continuous Integration
- Metaphor
- Small releases
- Simple Design
- Customer tests

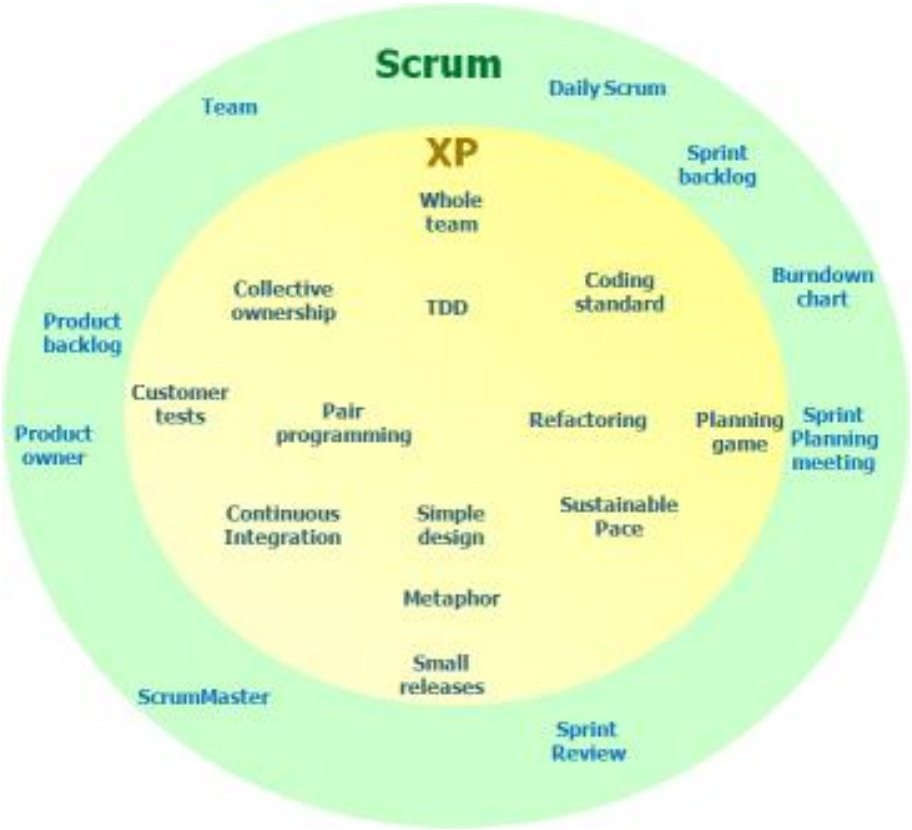


Complete Agile Process

Scrum + XP

Overview

How Scrum and XP can work together



Overview

- **XP**
  - **General concepts**
- Specific XP concepts
  - Pair programming
  - Test Driven Development
  - Continuous Integration

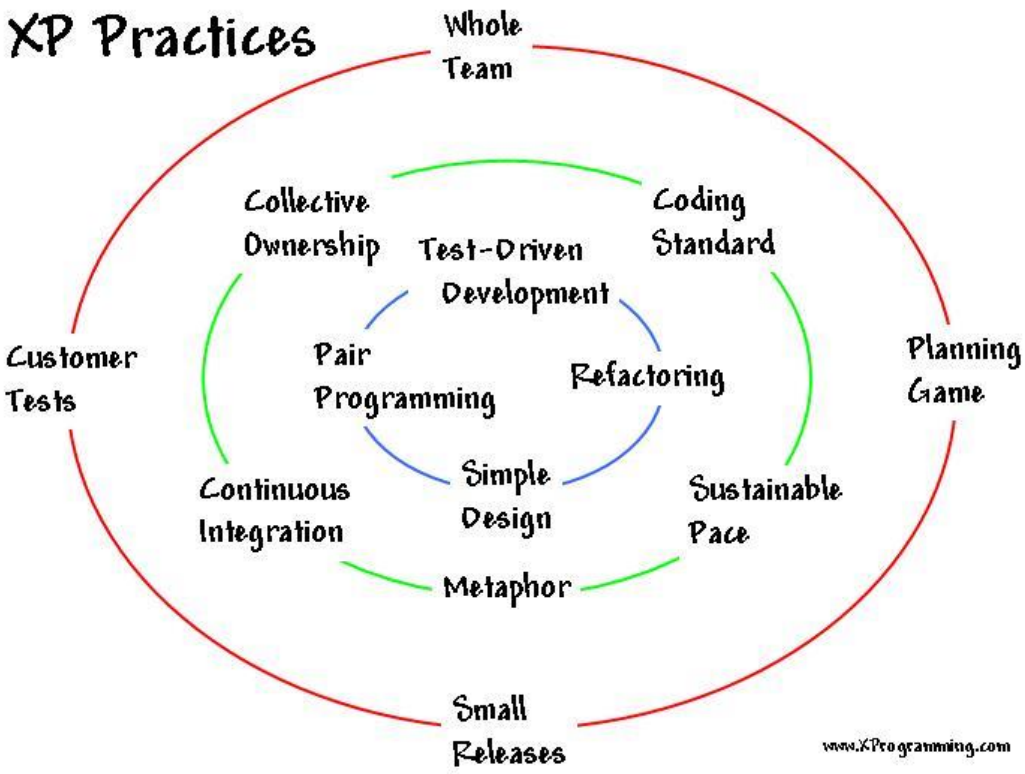
Principles of XP

Kent Beck

- **Communication**
  - Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team. XP stresses the importance of the appropriate kind of communication – face to face discussion with the aid of a white board or other drawing mechanism.
- **Simplicity**
  - Simplicity means "what is the simplest thing that will work?" The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means address only the requirements that you know about; don't try to predict the future.
- **Feedback**
  - Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design. Your team builds something, gathers feedback on your design and implementation, and then adjust your product going forward.
- **Courage**
  - Kent Beck defined courage as "effective action in the face of fear" (Extreme Programming Explained P. 20). This definition shows a preference for action based on other principles so that the results aren't harmful to the team. You need courage to raise organizational issues that reduce your team's effectiveness. You need courage to stop doing something that doesn't work and try something else. You need courage to accept and act on feedback, even when it's difficult to accept.
- **Respect**
  - The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and to work together to identify simple designs and solutions.

Source:https://www.agilealliance.org/glossary/xp

Practices of XP



Further Reading and Descriptions

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Whole Team

- All the contributors to an XP project sit together, members of one team. This team must include a business representative (Product Owner) – the "Customer" – who provides the requirements, sets the priorities, and steers the project.

https://ronjeffries.com/xprog/what-is-extreme-programming/#whole

Planning Game

- XP planning addresses two key questions in software development: predicting what will be accomplished by the due date, and determining what to do next.
- *Release Planning* is a practice where the Customer presents the desired features to the programmers, and the programmers estimate their difficulty.
- *Iteration Planning* is the practice whereby the team is given direction every couple of weeks. (Sprints)

https://ronjeffries.com/xprog/what-is-extreme-programming

Customer Tests

- As part of presenting each desired feature, the XP Customer defines one or more automated acceptance tests to show that the feature is working. The team builds these tests and uses them to prove to themselves, and to the customer, that the feature is implemented correctly.

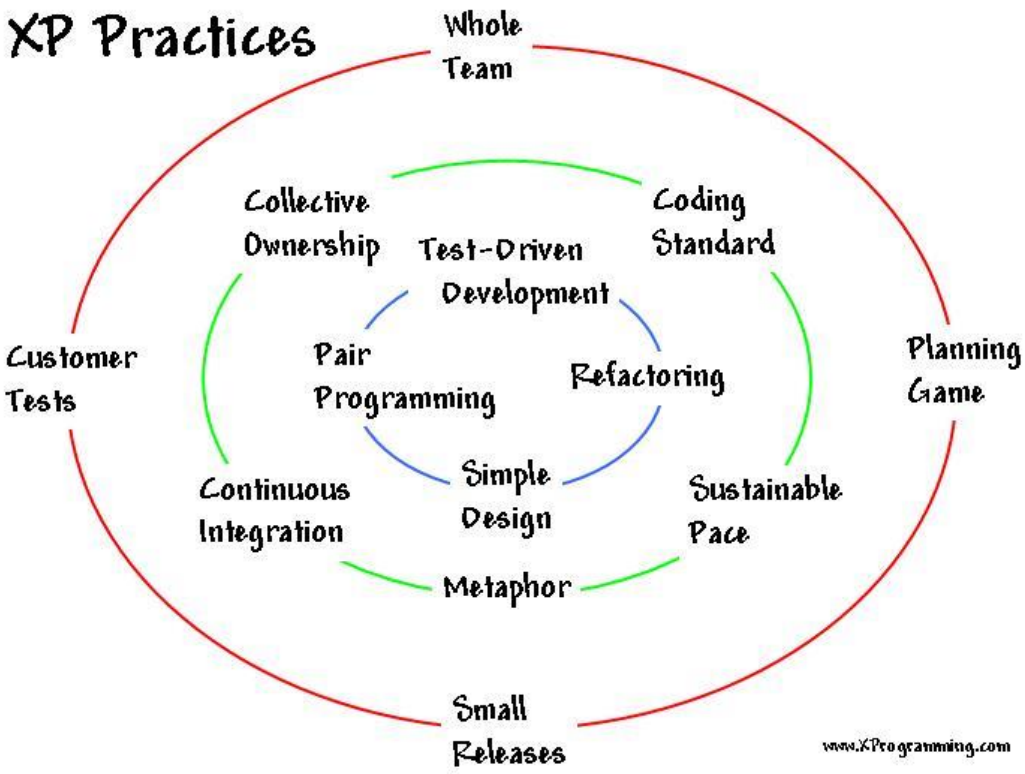
https://ronjeffries.com/xprog/what-is-extreme-programming/#whole

Small Releases

- XP teams practice small releases in two important ways:
  - First, the team releases running, tested software, delivering business value chosen by the Customer, every iteration.
  - Second, XP teams release to their end users frequently as well.

https://ronjeffries.com/xprog/what-is-extreme-programming/#whole

Practices of XP



Further Reading and Descriptions

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Coding standards

- XP teams follow a common coding standard, so that all the code in the system looks as if it was written by a single – very competent – individual. The specifics of the standard are not important: what is important is that all the code looks familiar, in support of collective ownership.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Collective Ownership

- On an Extreme Programming project, any pair of programmers can improve any code at any time. This means that all code gets the benefit of many people's attention, which increases code quality and reduces defects.

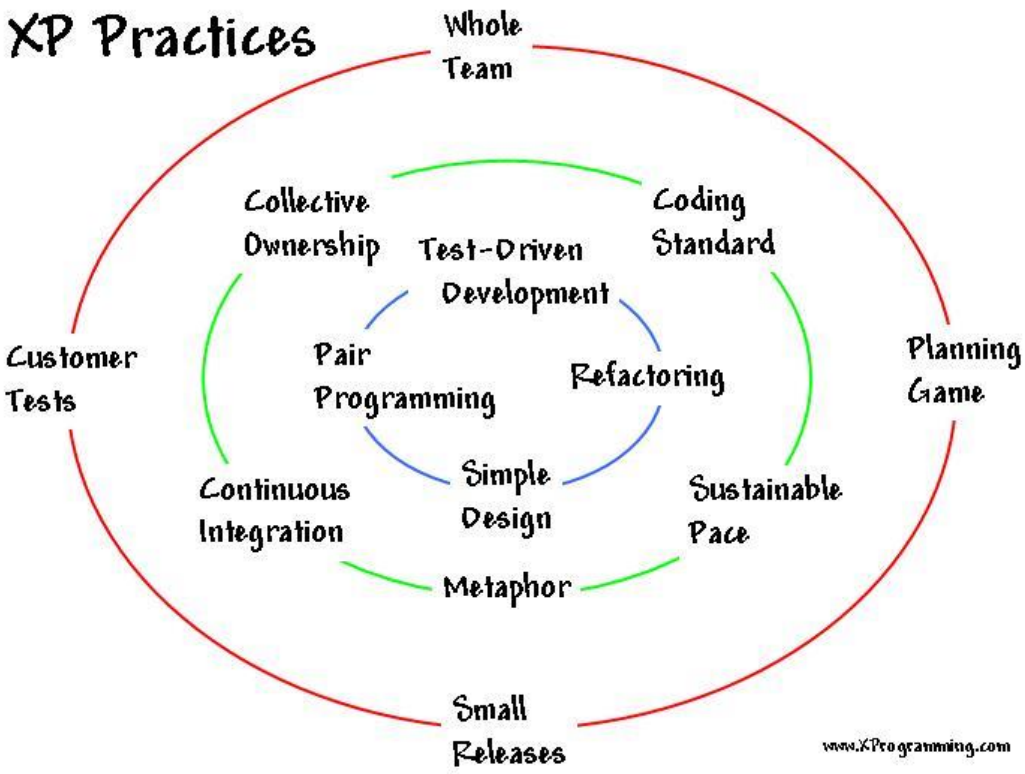
http://ronjeffries.com/xprog/what-is-extreme-programming/.

Metaphor

- Extreme Programming teams develop a common vision of how the program works, which we call the "metaphor". At its best, the metaphor is a simple evocative description of how the program works, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Practices of XP



Further Reading and Descriptions

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Refactoring

- The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "co*hesion*" of the code, while lowering the "co*upling*". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Simple Design

- XP teams build software to a simple but always adequate design. They start simple, and through testing and design improvement, they keep it that way. An XP team keeps the design exactly suited for the current functionality of the system. There is no wasted motion, and the software is always ready for what's next.

http://ronjeffries.com/xprog/what-is-extreme-programming/.

Overview

- ~~XP~~
  - ~~General concepts~~

- Specific XP concepts
  - Pair programming
  - Test Driven Development
  - Continuous Integration

Pair Programming

"You'll never work alone"

Not without precedent

Pair programming

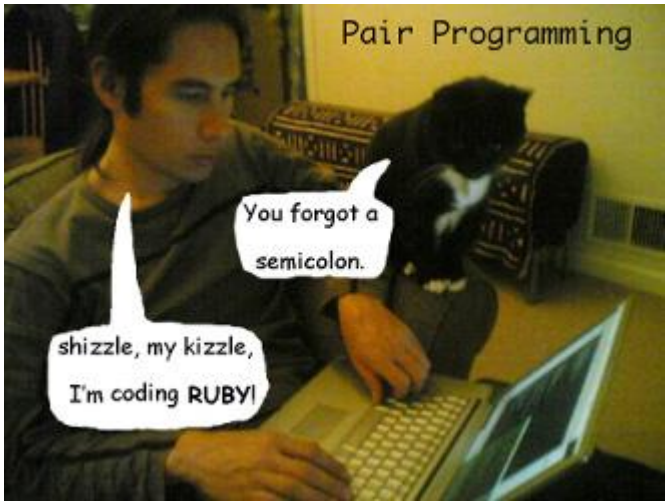
- Two developers working on the same task as a team
- One controls the keyboard one sits looking over their shoulder
- Driver
  - This person writes the code
- Navigator
  - This person reviews each line as it is typed

Advantages

- Higher code quality
  - Fewer bugs, code rewrites, integrations problems
- Expert novice pairing can help the novice to learn about the system and best practices
- Tends to produce more design alternatives and catches design defects earlier

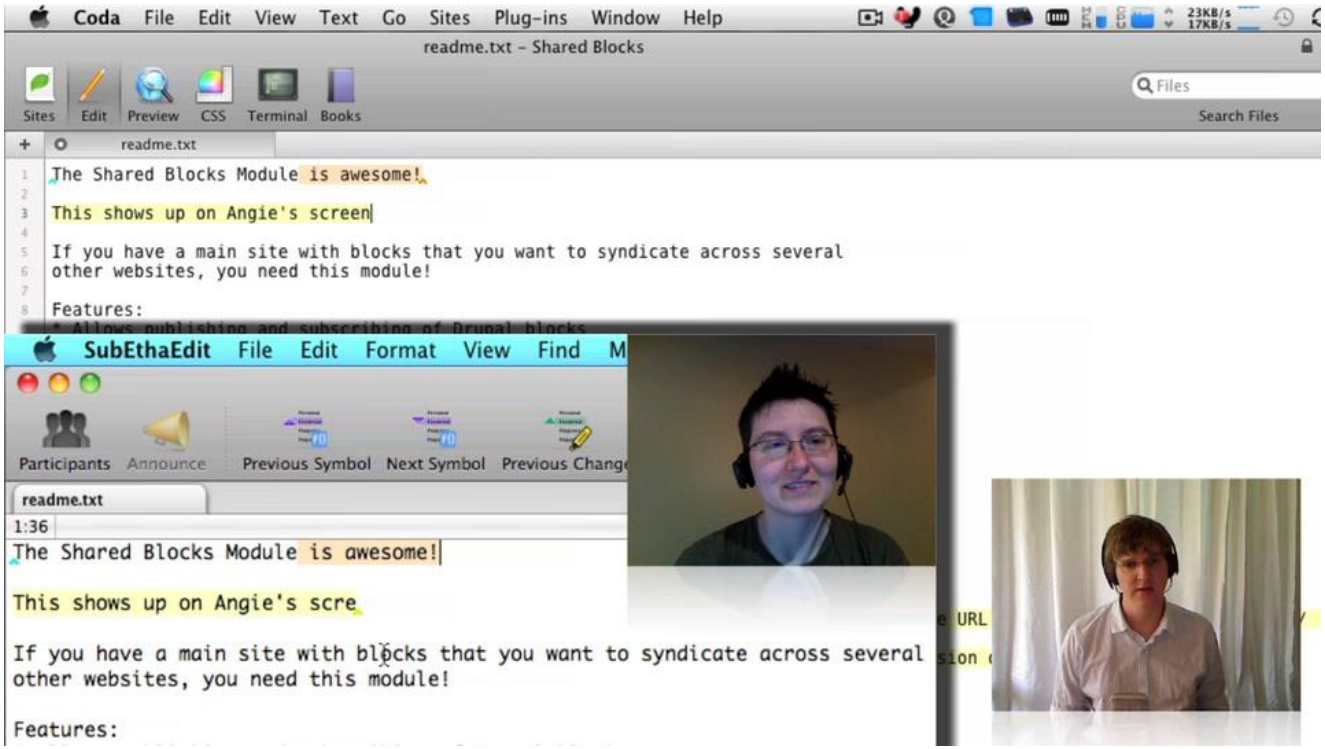
Disadvantages

- There is a high probability of disengagement
- "Watch the master" phenomenon
- Working relationship needs to be good
- Hard sell to management
  - Two people working on 1 feature



Remote pair programming

- Using communications technology
  - Screen sharing
  - IM clients, VOIP etc

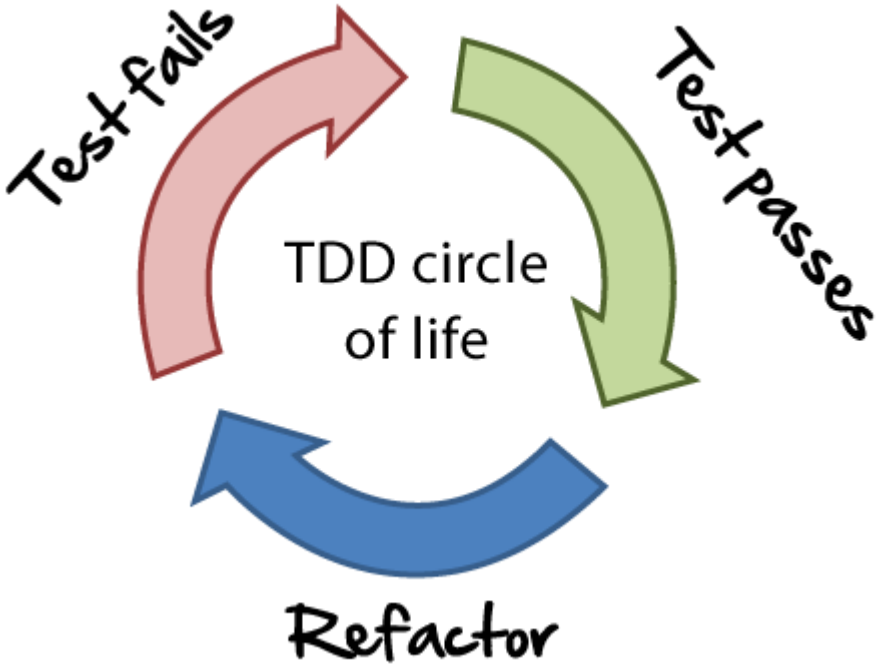


Overview

- ~~XP~~
  - ~~General concepts~~

- Specific XP concepts
  - ~~Pair programming~~
  - **Test Driven Development**
  - Continuous Integration

Test Driven Development (TDD)



TDD Cycle

- Create a test
  - Each new feature requires a test
- Run all tests
  - Make sure the new test fails
- Write the code
  - Doesn't have to be perfect, just pass the test
- Run all tests
  - If all tests pass, requirements are met
- Refactor code
  - TDD can result in duplication, this should be removed
- Repeat the process

Principles of TDD

- Never write new functionality without a failing test
- Continually make small incremental changes
- Keep the system running at all times
  - No one can make a change that breaks the system
  - Failures must be addressed immediately

Advantages

- Discourages “gold plating” of implementation
- Forces the developer to specify an end criteria
- Encourages loose coupling

Disadvantages

- Big time investment
- Complexity in writing appropriate test cases
- Design changes
- Mock code to pass tests
- Customer may not wish to get involved in creating acceptance tests

Interesting - IBM Study

- Study carried out by IBM focussed on a team that had been practising TDD for 5 years and delivered 10 releases of a software product

- Quality was the big winner, much improved, fewer defects/bugs etc

- Productivity did **decrease** but not dramatically

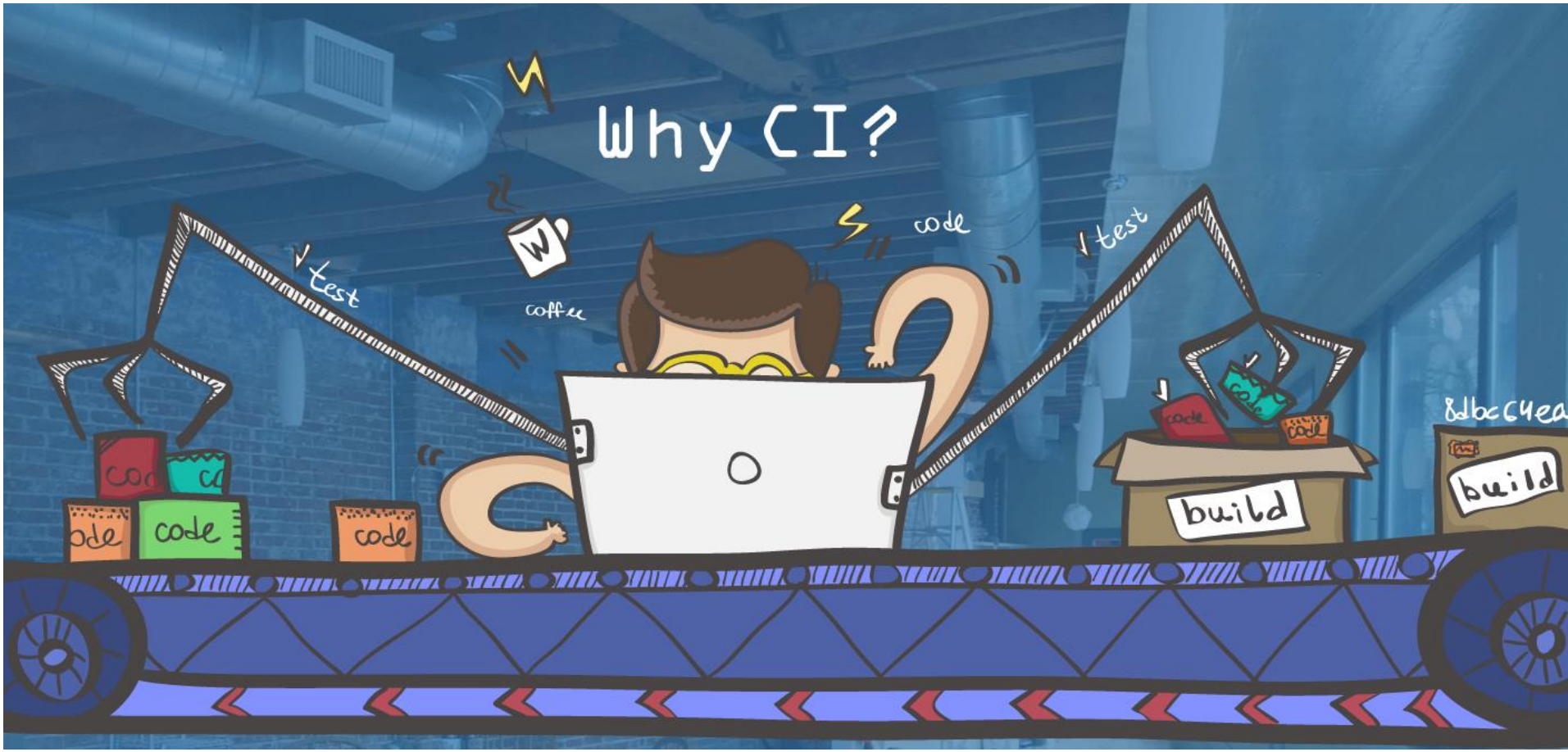
Sanchez, J., Laurie Williams, and E. Michael Maximilien. "A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry." Proc. Agile. 2007.

Learning objectives

- ~~XP~~
  - ~~General concepts~~

- Specific XP concepts
  - ~~Pair programming~~
  - ~~Test Driven Development~~
  - **Continuous Integration**

Continuous Integration

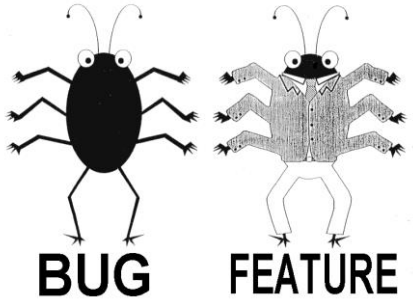


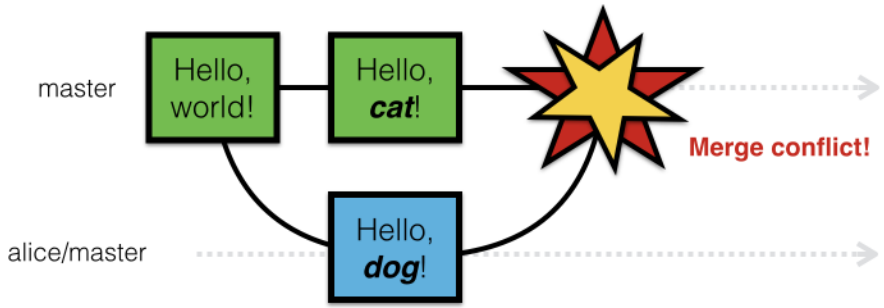
Continuous Integration (CI)

- CI is a practice where members integrate their changes frequently.
  - Often daily
- Each integration is verified by an automated build including tests to detect integration errors as early as possible.
  - Often upon commit, builds are run to make sure everything is okay

Development before CI

- Lots of bugs
- Infrequent commits
- Difficult integration
- Infrequent releases
- Testing happens late

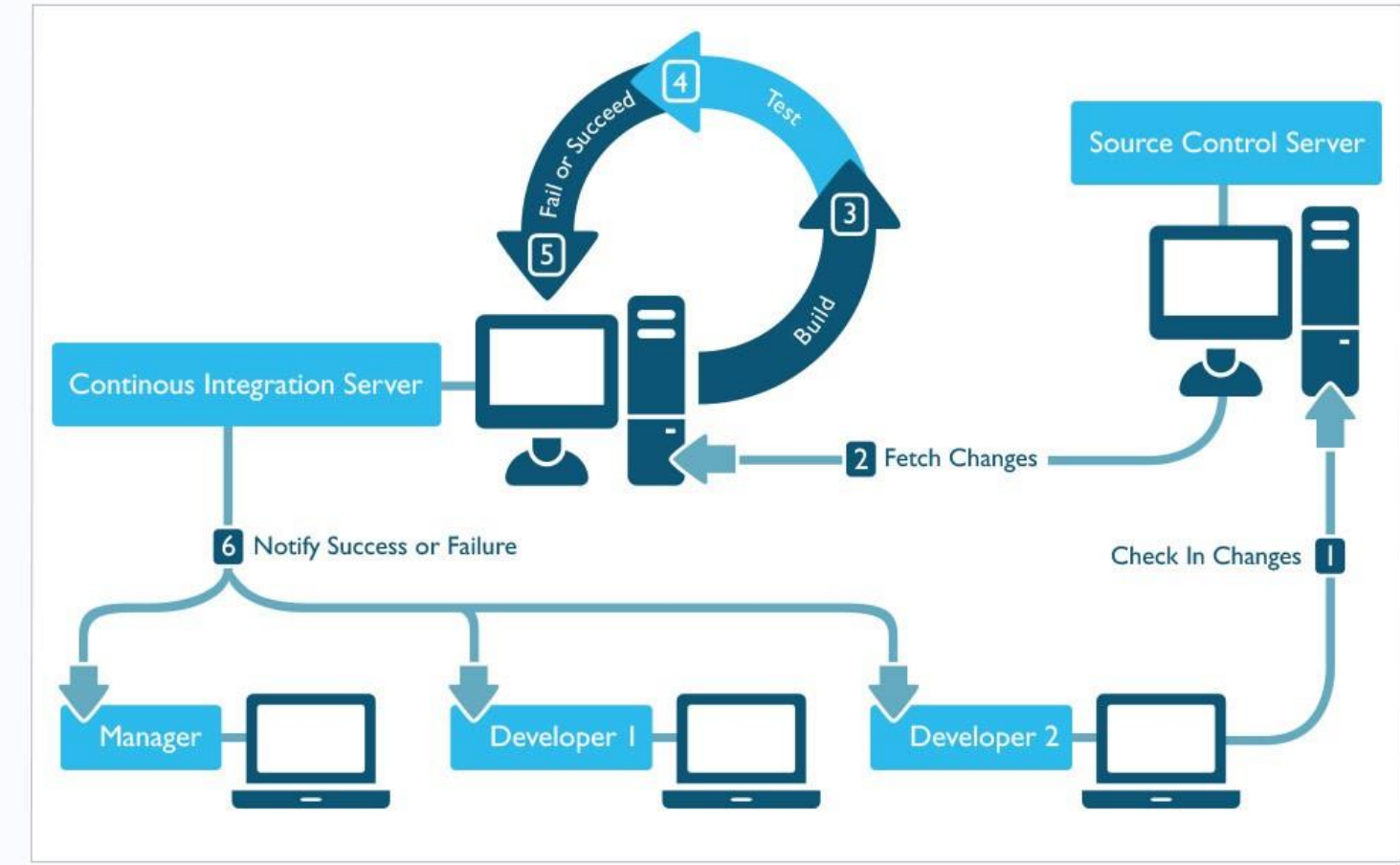




Benefits of CI

- Fail early/fast
  - Detect problems as early as possible
- Facilitates continuous deployments
  - Deploying every good build live to production
- Enables automated testing
  - Tests are run during the build process

Overview



https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

Drawbacks of using CI

- Initial setup required
  - Can take a couple of weeks to get it running properly within an organisation
- Excellent tests must be developed
  - CI will run all the automated tests but this requires substantial up front development effort.

Popular CI software

CODESHIP

circleci

Bamboo

Solano Labs

snap

semaphore

AppVeyor

drone.io

SETTING UP OUR DEVELOPMENT ENVIRONMENTS

Dr. Enda Barrett – Enda.Barrett@nuigalway.ie

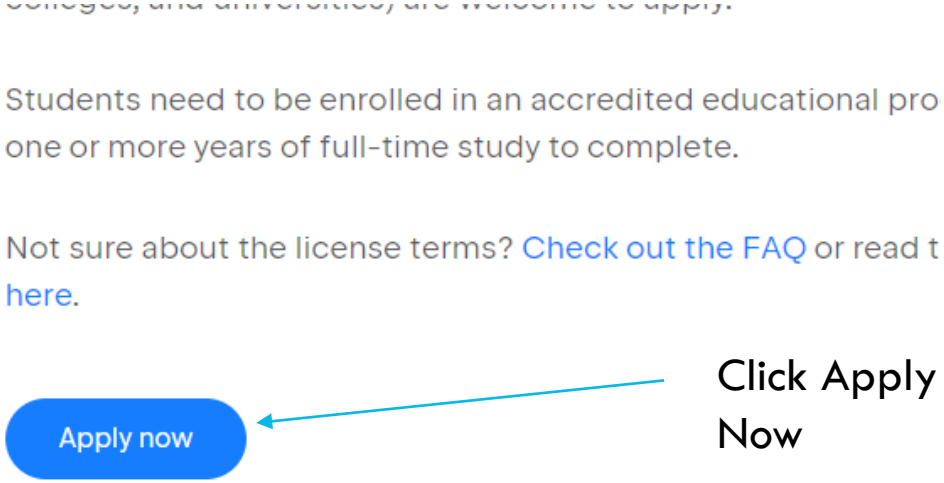
Ollscoil na Gaillimhe
University of Galway

Getting things set up

- Install an IDE
- Install software (NodeJS runtime environment)
- Quick overview of Firebase
- Claim cloud credits
- Create a new Google Account using your University email address
- Create a new Firebase project
- Create a new local project
- Using the Command Line
- Install Firebase CLI
- Initialise the projects to link with the cloud
- Deploy to the cloud

WebStorm - IntelliJ

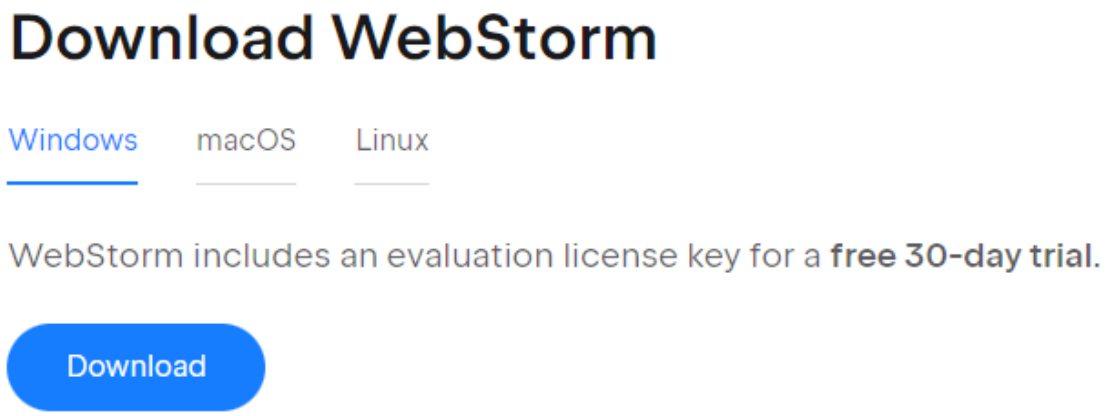
- Navigate to
  - https://www.jetbrains.com/community/education/#students



- Fill in the form details and you will get a 1 year license
  - There are lots of alternatives too if you prefer to use something else

Download WebStorm

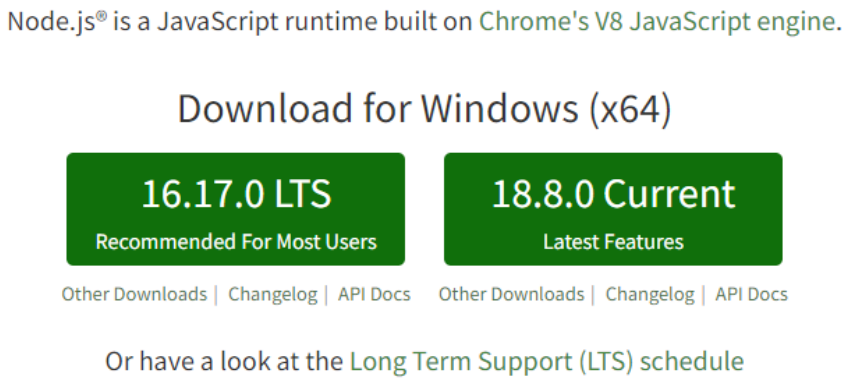
- https://www.jetbrains.com/webstorm/download/#section=windows



Install NodeJS

- Navigate to https://nodejs.org/en/

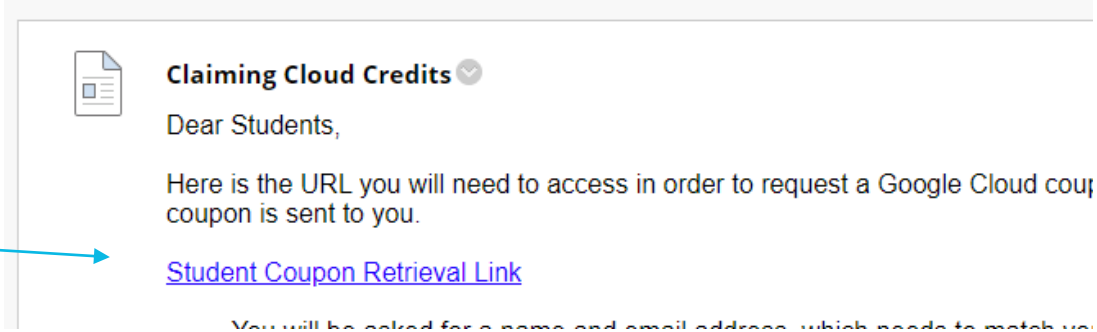
- Install the latest LTS (Long Term Support) version of NodeJS



Cloud Credits

- Google have kindly provided each student with $25 in cloud credits to use on their Firebase platform
- On Blackboard under Week 2, you will see this post

Click the link

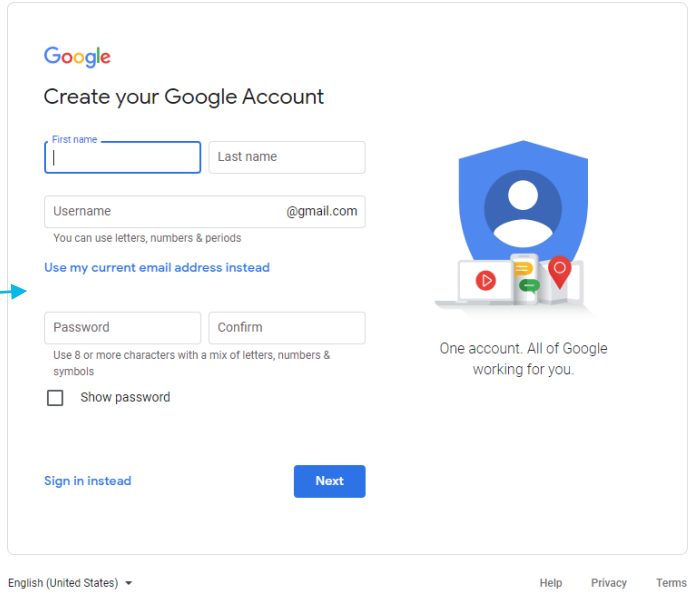


- You must use your university email address when claiming the credits

Create a Google Account

- Perform a quick search for Google Account Creation

Click on use my current email address



- Use your **university email address** for this. Same one you claimed the credits with!

CREATING A NEW FIREBASE PROJECT

Dr. Enda Barrett – Enda.Barrett@nuigalway.ie

Ollscoil na Gaillimhe
University of Galway

Firebase – Quick overview

- Originally Firebase was a (Mobile) Backend as a Service offer which is a subset of a Platform as a Service
- Whilst an IaaS provider does help you to move away from on-prem hosting, you do have a good bit of setup. Install a database on the virtual server, varying software packages, even an OS! You have to keep it updated and secure etc.
- With a PaaS offering such as Firebase you can focus on coding!
- It provides a suite of services
  - Storage
  - Database
  - Authentication
  - Machine Learning
  - Functions
  - Messaging
  - Hosting

Firebase

My sample project

Project Overview

Product categories

Build
- Authentication
- App Check
- Firestore Database
- Realtime Database
- Extensions
- Storage
- Hosting
- Functions
- Machine Learning
- Remote Config
- Cloud Messaging

Release & Monitor

Analytics

Engage

All products

Spark
No-cost $0/month
Upgrade

Receive email updates about new Firebase features, research, and events Sign up

My sample project

Spark plan

Get started by adding Firebase to your app

iOS+

Add an app to get started

Store and sync app data in milliseconds

Authentication

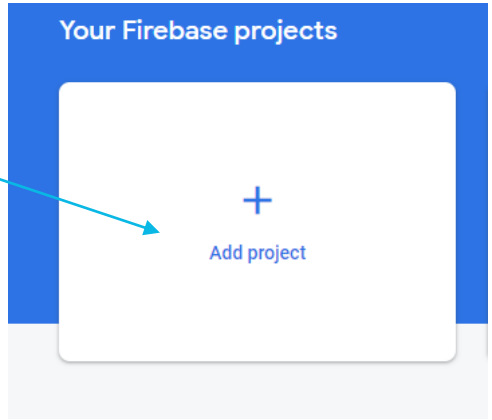
Authenticate and manage users

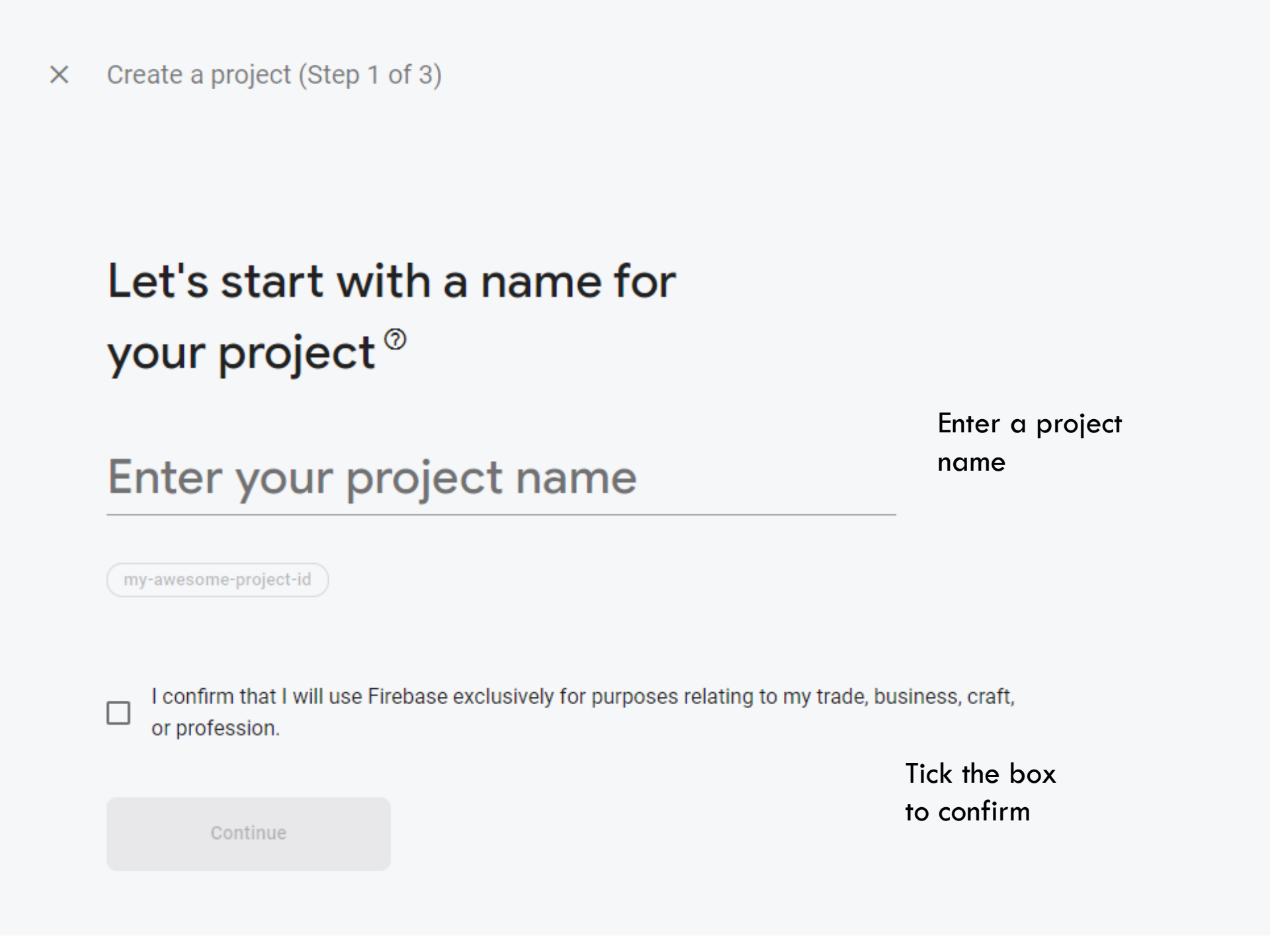
Cloud Firestore

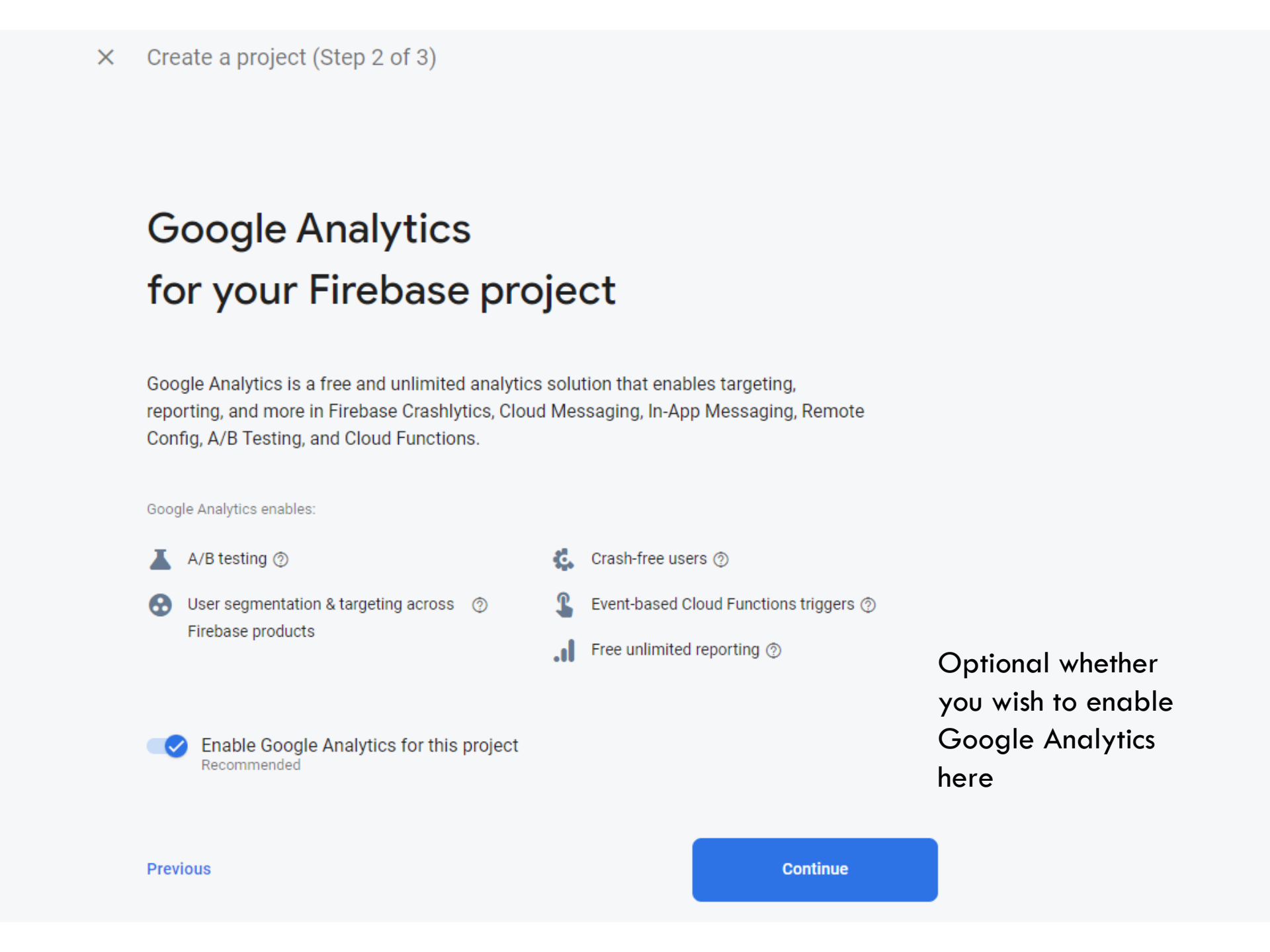
Realtime updates, powerful queries, and automatic scaling

Firebase project

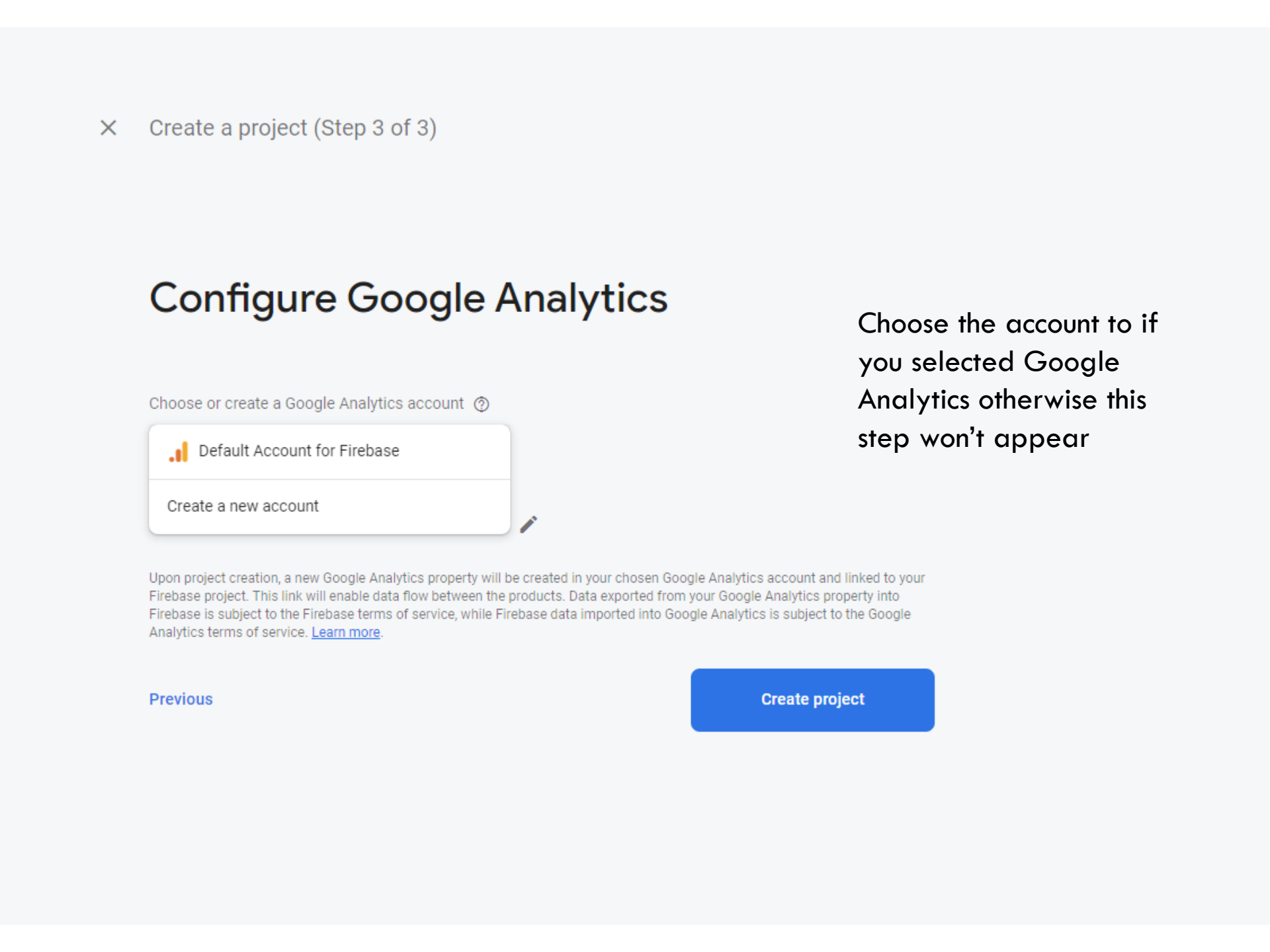
- We will need to create Firebase projects for our apps
- Login using your Google Account created with your university email address
- Create a new project by clicking here




Create a project (Step 1 of 3)
Let's start with a name for your project
Enter your project name
my-awesome-project-id
I confirm that I will use Firebase exclusively for purposes relating to my trade, business, craft, or profession.
Continue
Enter a project name
Tick the box to confirm



Optional whether you wish to enable Google Analytics here



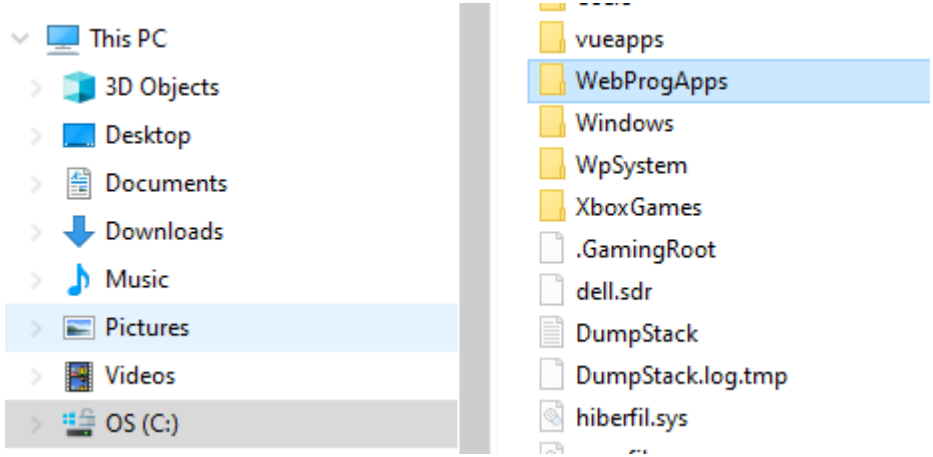
Choose the account to if you selected Google Analytics otherwise this step won't appear

Creating a local directory

- In order to write our applications we will code them locally on our computers, save our code files and then push them to the cloud.
- In order to work locally we need to create a folder to store our code.
- It is best to place this on root the of your machine or some place that is easy to access from a **command line** perspective.
- In windows create a folder on C:\WebProgApps

May look something like this

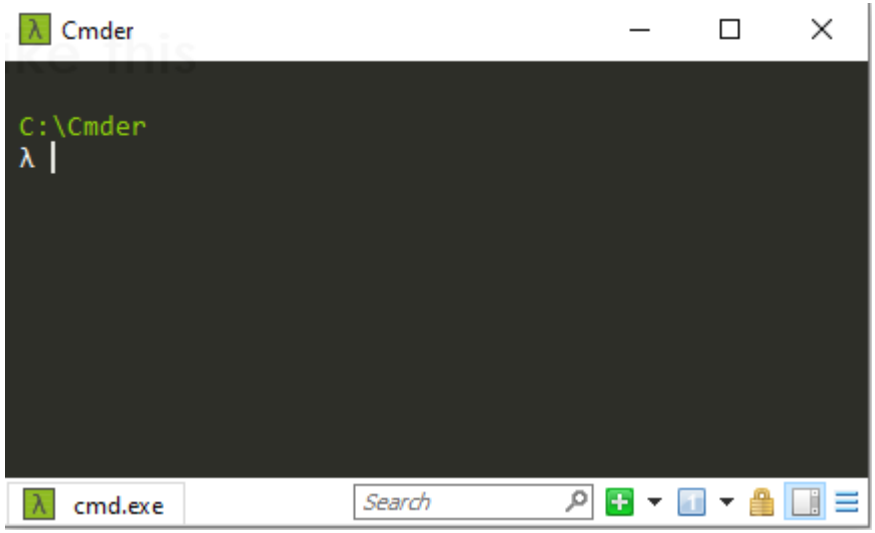
- If you have a windows machine you can add a folder like this.



- On a MAC you can use Finder to the same, if using Linux you already know this!

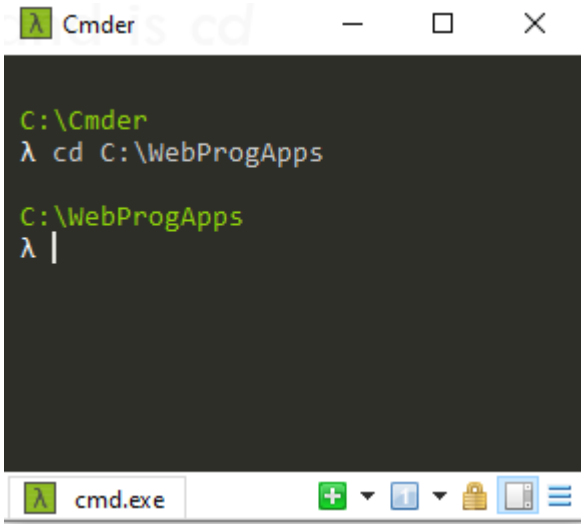
What is the Command Line Interface

- A mechanism through which you can execute commands to the OS. It's how we used to operate computers before their was a nice GUI.
- Depending on your OS (Windows, MAC or Linux) you search for Command Prompt(Windows), Terminal (MAC or Linux).
- It will look something like this



Navigate to the root

- Depending on the OS the default directory that the command line will begin with once opened can be different.
- We wish to navigate to the folder we have created on the root to get it set up for development.
- To change directory the command is *cd*
- Type cd C:\WebProgApps and press Enter on the keyboard

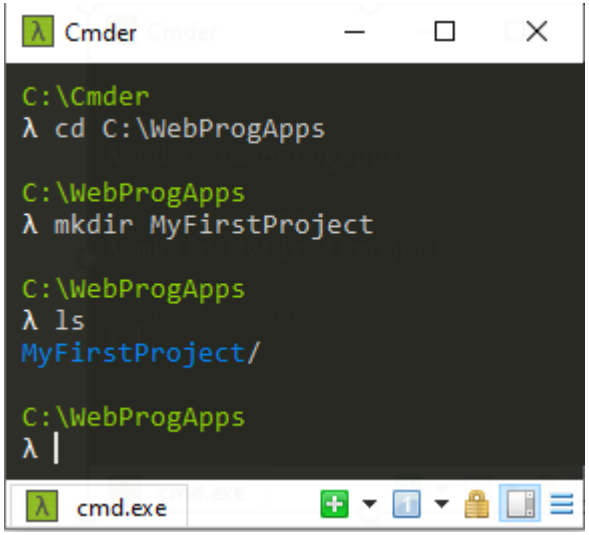


Creating a directory

- You can avoid having to use the GUI to create a directory and use the mkdir command instead.
- If you called your project on Firebase MyFirstProject then we can use the same name for the local folder which will contain the code for this project
- Type mkdir MyFirstProject

- Check the folder it should be there using

  dir (windows) or ls (MAC)



Install the Firebase CLI

- Firebase CLI - https://github.com/firebase/firebase-tools
- A suite of tools for managing, visualising and deploying Firebase projects
- Install it onto our machines and then we can use it to configure our projects, link them to the cloud backend and then deploy them onto the web!
- Two ways to install it, you can use the standalone binaries which are available in the link below or you can use ***npm*** *(Node Package Manager)*

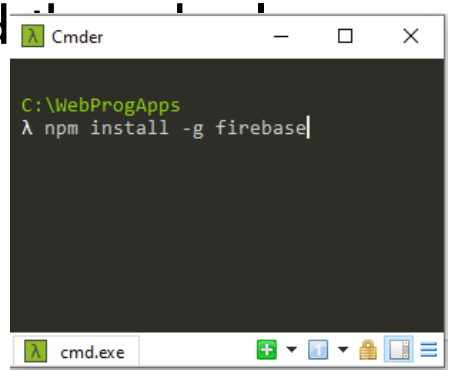
https://firebase.google.com/docs/cli

NPM – Node Package Manager

- Node Package Manager, is the default package manager for the NodeJS runtime.
  - https://www.npmjs.com/
- It contains thousands of Open Source projects containing code which we call packages.
- We can install packages onto our machine by using the NPM command. These can be executed as standalone programs on our machines (Firebase CLI) or included in our own programs that we write.
- When we installed NodeJS (back in Slide 4), NPM was also included. Thus we can use it by typing the command npm

Install Firebase

- Step 1 – Install the Firebase CLI
  - npm install -g firebase
  - npm install -g firebase-tools
  - The g argument is for global – all projects on the machine will have access to the CLI
  - If you are using a MAC you will need to include sudo
    - sudo npm install –g firebase
    - sudo npm install –g firebase-tools
- Once you have typed in the command press Enter on the keyboard
- Install it onto our machines and then we can use it to configure our projects, link them to the cloud backend and [illegible] them onto the web!

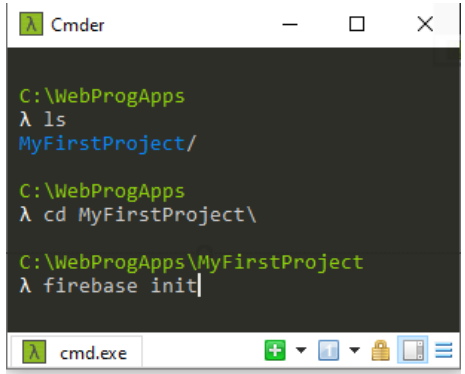


Logging into Firebase

- In order to perform the next sequence of steps we need to log into Firebase using the command line.
- Enter the command firebase login and hit Enter
- This will open up a web browser where you can login using your Google Account created with your university email address.

Initialise our project folder

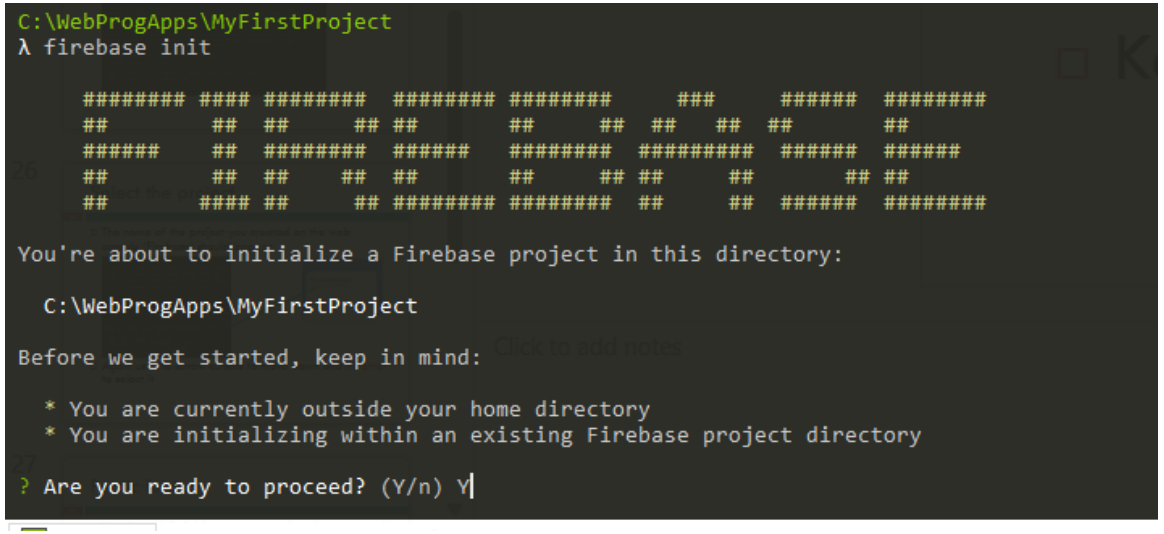
- In order to link our local project folders with the online cloud project we need to initialise it.
- Make sure you are in the correct directory on the command line



- Enter the command firebase init and press Enter

Sequence of init steps

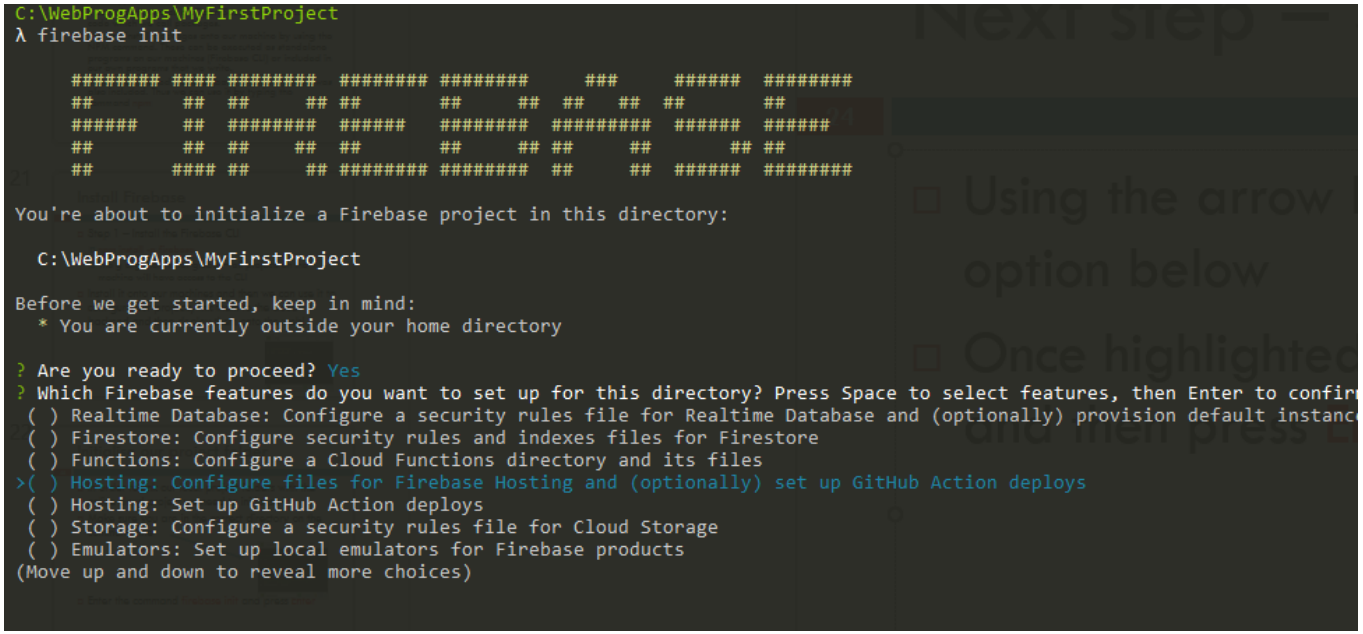
- If everything has worked correctly you should now see the following



- Key in Y and pres Enter

Next step – Setup hosting

- Using the arrow keys move down to the highlighted option below
- Once highlighted select it using the spacebar key and then press Enter



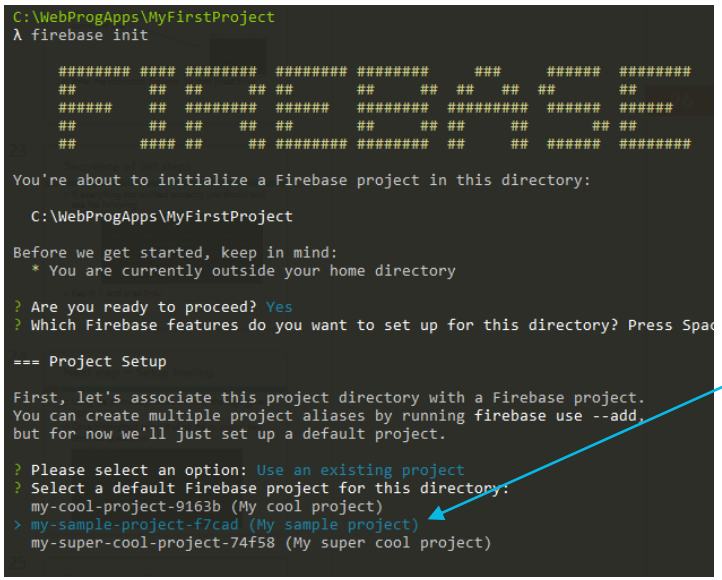
Select an existing project

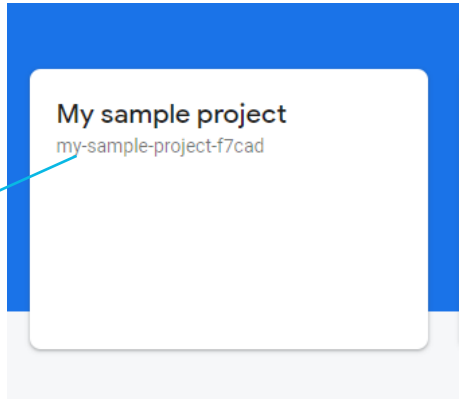
- Select use an existing project by hitting Enter

```
  C:\WebProgApps\MyFirstProject

Before we get started, keep in mind:
  * You are currently outside your home directory

? Are you ready to proceed? Yes
? Which Firebase features do you want to set up for this directory? Press Space to

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: (Use arrow keys)
> Use an existing project
  Create a new project
  Add Firebase to an existing Google Cloud Platform project
  Don't set up a default project
```

Select the project

- The name of the project you created on the web console (Firebase) should appear





- Again use the arrow cursors to move down and hit Enter to select it

Public folder

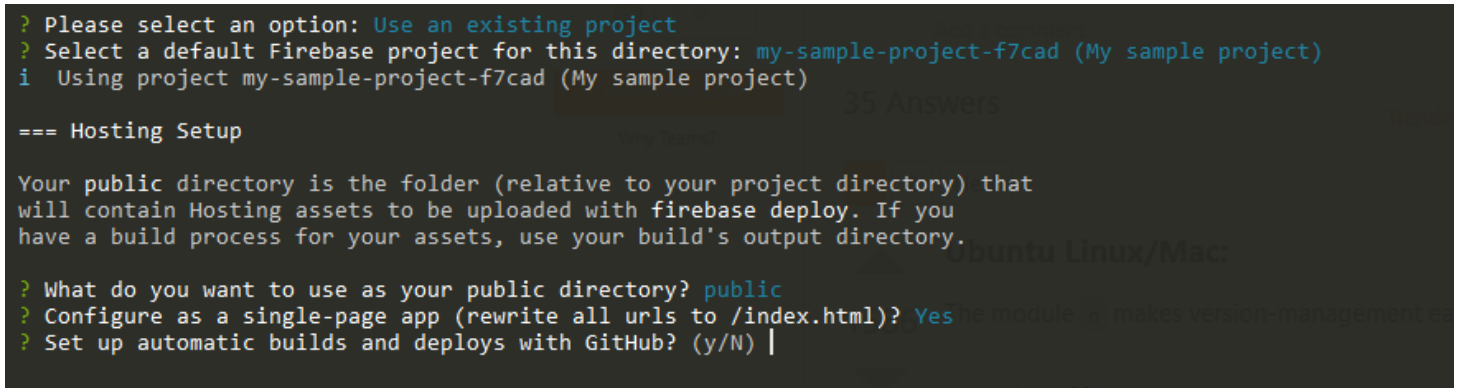
- Use the public folder as prompted on the command line. This is where we will place all of frontend code.

```
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: my-sample-project-f7cad (My sample project)
i  Using project my-sample-project-f7cad (My sample project)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? (public)
```

- Press the Enter key on your keyboard

URL rewrite

- The next option is a technical setting which configures the app as a Single Page App (SPA) where requests are directed at index.html
- Select Y here and press Enter



Automatic builds and deploys

- There is no need to setup automatic builds and deploys at that this point

- Select N for this and press Enter

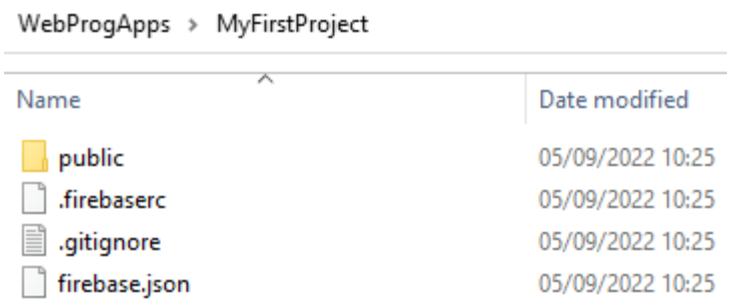
```
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: my-sample-project-f7cad (My sample project)
i  Using project my-sample-project-f7cad (My sample project)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? (y/N) N
```

Firebase initialisation should be complete


```
C:\WebProgApps\MyFirstProject
λ firebase init

     ######## #### ########  ######## ########     ###     ######  ########
     ##        ##  ##     ## ##       ##     ##  ##   ##  ##       ##
     ######    ##  ########  ######   ########  #########  ######  ######
     ##        ##  ##    ##  ##       ##     ## ##     ##       ## ##
     ##       #### ##     ## ######## ########  ##     ##  ######  ########

You're about to initialize a Firebase project in this directory:

  C:\WebProgApps\MyFirstProject

Before we get started, keep in mind:
  * You are currently outside your home directory

? Are you ready to proceed? Yes
? Which Firebase features do you want to set up for this directory? Press Space to select features, th

=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: Use an existing project
? Select a default Firebase project for this directory: my-sample-project-f7cad (My sample project)
i  Using project my-sample-project-f7cad (My sample project)

=== Hosting Setup

Your public directory is the folder (relative to your project directory) that
will contain Hosting assets to be uploaded with firebase deploy. If you
have a build process for your assets, use your build's output directory.

? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? No
+  Wrote public/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...
i  Writing gitignore file to .gitignore...

+  Firebase initialization complete!

C:\WebProgApps\MyFirstProject
λ
```

Files in the folder

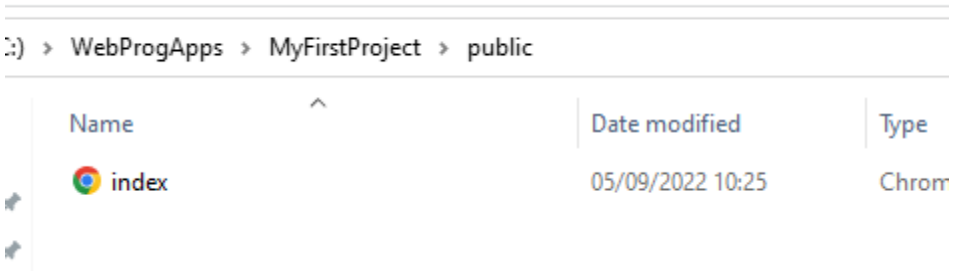
- If you look in your app folder on your machine you should now see a bunch of new files have been created, a public folder and an index.html file within the public folder



Deploying to the cloud

- The final step is to deploy to the cloud.

- We haven't written any code yet or created our first web page, but during the init process Firebase created a page index.html which you can find in the public folder.



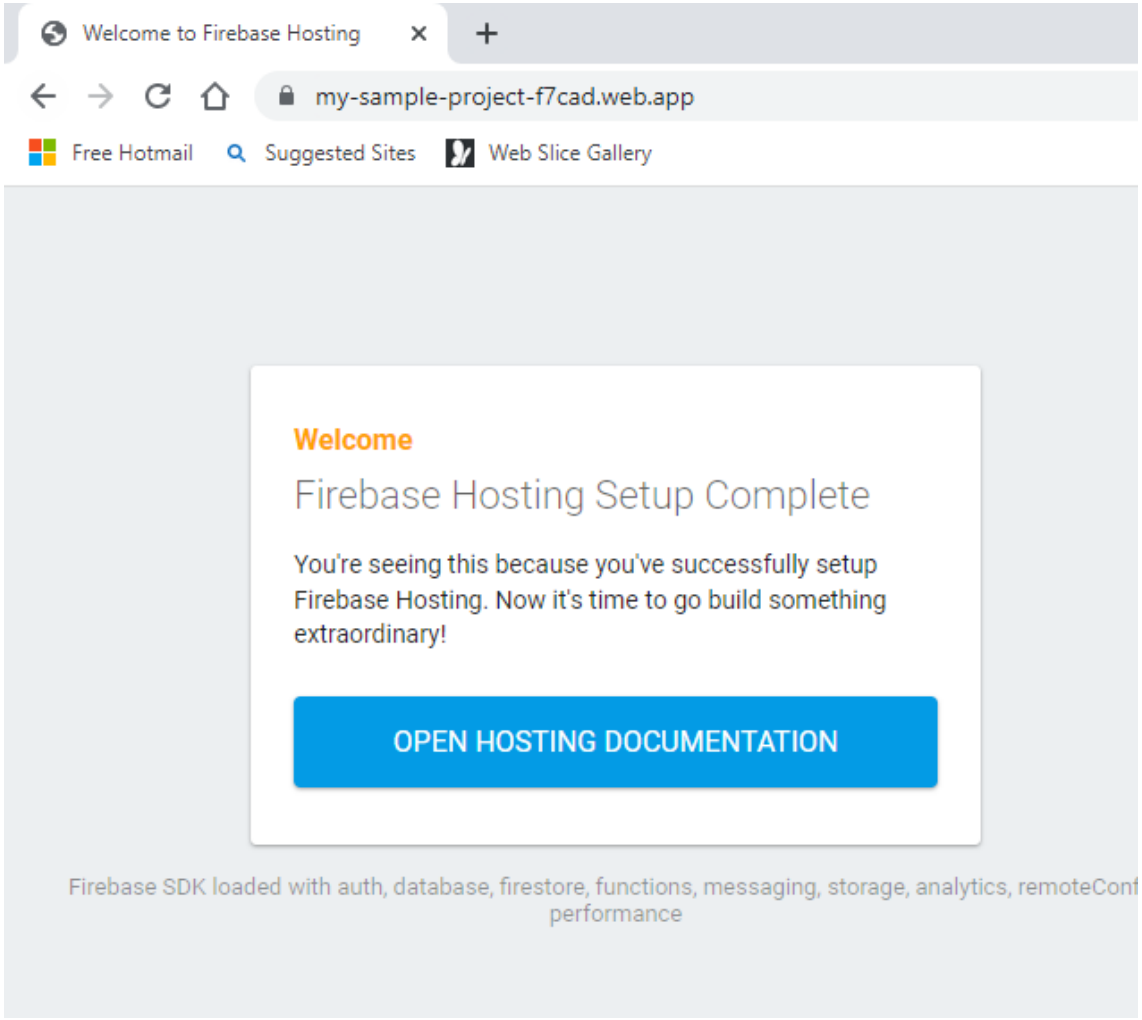
Firebase deploy

- The command to deploy our apps to the cloud is
  - firebase deploy

```
C:\WebProgApps\MyFirstProject
λ firebase deploy

=== Deploying to 'my-sample-project-f7cad'...

i  deploying hosting
i  hosting[my-sample-project-f7cad]: beginning deploy...
i  hosting[my-sample-project-f7cad]: found 1 files in public
+  hosting[my-sample-project-f7cad]: file upload complete
i  hosting[my-sample-project-f7cad]: finalizing version...
+  hosting[my-sample-project-f7cad]: version finalized
i  hosting[my-sample-project-f7cad]: releasing new version...
+  hosting[my-sample-project-f7cad]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/my-sample-project-f7cad/overview
Hosting URL: https://my-sample-project-f7cad.web.app

C:\WebProgApps\MyFirstProject
λ
```

Copy and paste the hosting URL and pop it into the address bar of your browser

It works!


Welcome to Firebase Hosting
my-sample-project-f7cad.web.app
Free Hotmail
Suggested Sites
Web Slice Gallery
Welcome
Firebase Hosting Setup Complete
You're seeing this because you've successfully setup Firebase Hosting. Now it's time to go build something extraordinary!
OPEN HOSTING DOCUMENTATION
Firebase SDK loaded with auth, database, firestore, functions, messaging, storage, analytics, remoteConf
performance

CREATING OUR FIRST WEB PAGE WITH HYPERTEXT MARKUP LANGUAGE (HTML)

Dr. Enda Barrett
Enda.Barrett@universityofgalway.ie

OÉ Gaillimh
NUI Galway

HTML

- Stands for Hyper Text Markup Language (HTML)
- Notation for describing document structure and formatting
- A html file has a .html or .htm extension
- It is rendered by a browser

Simple HTML Document

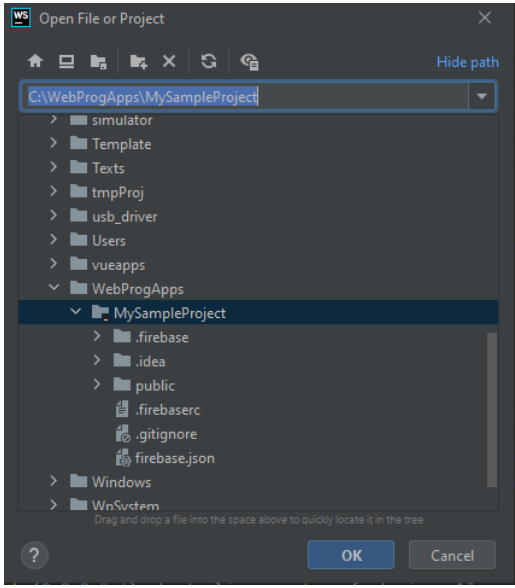
- Document starts with: `<html>`
- Document ends with: `</html>`
- Text between `<head>` and `</head>` is header information which is not rendered by the browser
  - `<title>` and `</title>` displays the title of the document

- Everything between `<body>` and `</body>` is rendered and displayed by the browser
- Contains actual text to display and tags defining its style, layout etc. plus additional elements e.g. images

Open Project in WebStorm

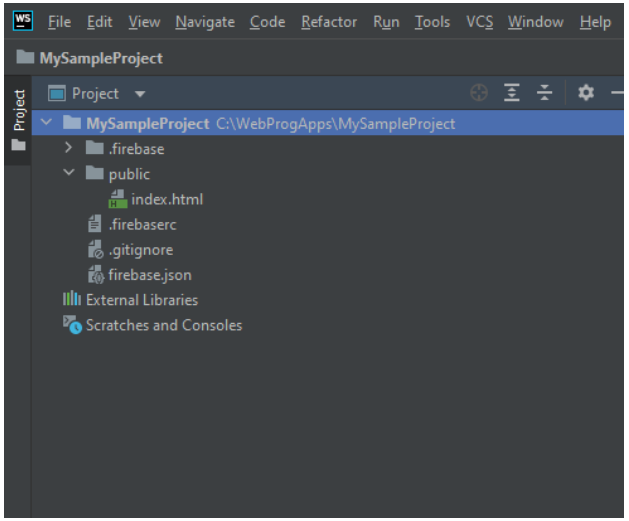
- Using WebStorm (or preferred IDE) open up the firebase project that you created in the previous lecture.

- In WebStorm select File>Open
- This will pop up a dialog where you can select your project (MySampleProject)



Once opened in WebStorm

- You will see a view (see screenshot below) where the project is on the left hand-side including the files and folders
- If you expand the public folder you will see the index.html file.
- Double click it to start editing
- This file is our first HTML page!



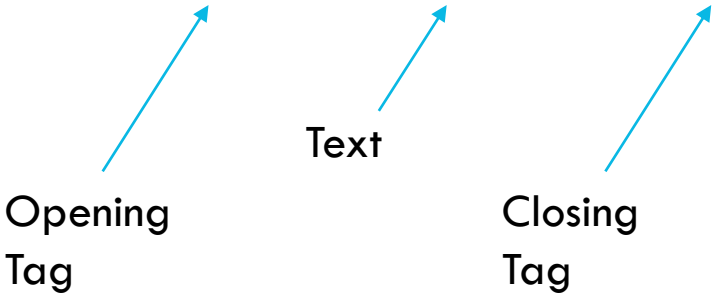
Exercise 1 - HTML – Hello World


```
<!DOCTYPE html>
<html>
<head>
  <title>Simple Page</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

- Delete all the existing markup from index.html
- Take the following markup and insert it into index.html
- Surround the “Hello World” text with a heading 1 tag – (search online).
- Save the file
- Deploy it to Firebase (firebase deploy) command line

HTML Tags

- HTML contains text to display and markup tags
  - <h1>Hello World</h1>



- The tags tell the browser how to display the contents of a page: colours, formats, positions, etc.

HTML Tags

- Tags denote markup **elements**
- Each tag is surrounded by angle brackets < >
- Tags normally come in pairs:
  - the **opening tag** and the **closing tag**
- Tags are **not** case sensitive
  - <html> and <HTML> are functionally the same
  - Recommended to use lowercase
- Text between the tags is the **element content**
- `<h1> ... </h1>` Section 1
- …
- `<h6> ... </h6>` Section 1.1.1.1.1.1

Tag Attributes

- Tags can have associated attributes (or properties) that provide extra information to the browser
- Attributes consist of **`name="value"`** pairs
- Attributes are always added to the **opening tag**
- For example, to colour the text of heading 1 tag to red:
  - *Set the attribute style to the value of **color:red***
  - `<h1 style="color:red"> Section 1 </h1>`

Section 1

Tag Attributes cont'd

- Attribute values normally enclosed in quotes
  - Double quotes are the most common
  - `style="color:red"`
  - Single quotes are also allowed
  - `style='color:red'`

Be careful with copy and paste

- NB: If the value contains one type of quote, then the other type should be used to enclose the value
  - `name="John's Place"`

Common Tags - Text

- Bold

  `<b>Bold Text</b>` **Bold Text**

- Italic

  `<i>Italic Text</i>` *Italic Text*

- Paragraph

  `<p>Text</p>` Defines a paragraph

Exercise 2

1. Extend your web page to include two paragraphs of dummy text
2. Italicise any words beginning with *s*
3. Bold any words beginning with **a**

- Text
- "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat."

- "Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Common Tags - Continued

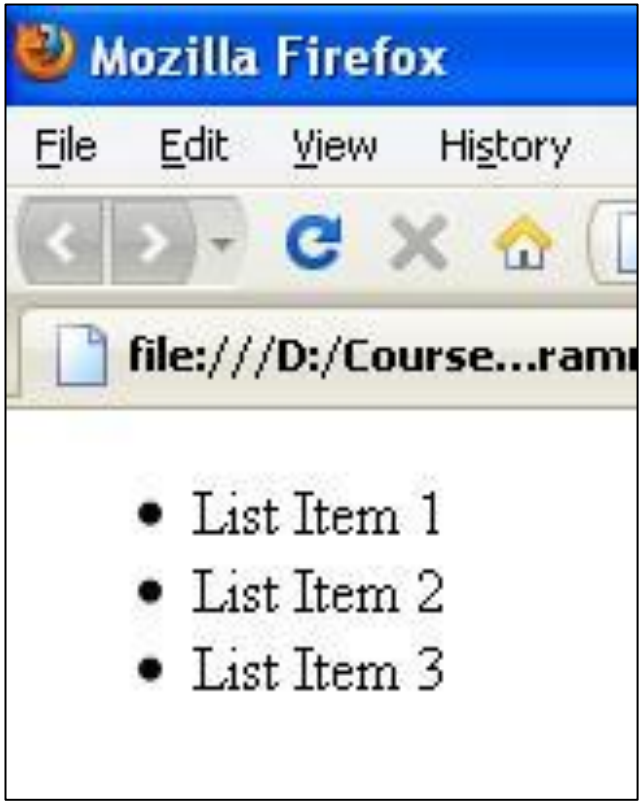
- Line Break
  - Forces a line break wherever it's placed
  - Putting a carriage-return into the HTML code will not produce a visible line break!
  - This
is a para
graph with line breaks
- Horizontal Rule <hr>
  - Can take width attribute <hr width="75%">
- Special Entities (start of a character reference)
  - Ampersand &
  - Copyright © ©

HTML Lists

- **U**nordered **L**ists `<ul> </ul>`
  - Each **L**ist **I**tem appears as Bullet `<li> </li>`
- **O**rdered **L**ists `<ol> </ol>`
  - Bullets replaced with numbers or letters
  - Type of order defined with type attribute
- **D**efinition **L**ists `<dl> </dl>`
  - Lists a **d**efinition **t**erm `<dt> </dt>`
  - and the **d**efinition **d**escription `<dd> </dd>`

Unordered Lists

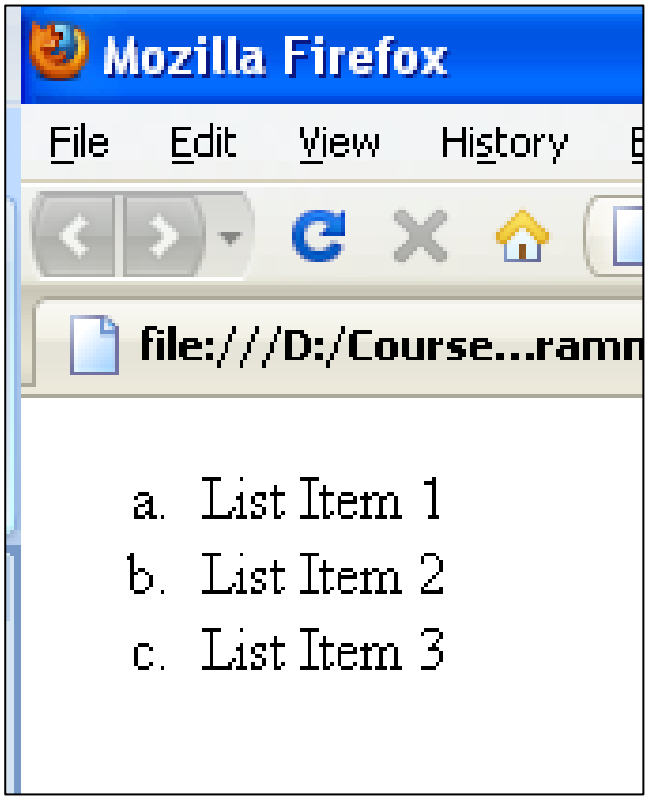

```
<ul>
        <li>List Item 1</li>
        <li>List Item 2</li>
        <li>List Item 3</li>
</ul>
```



- Bullet is default type
- Other types can be specified
  - "disc", "square", "circle", etc

Ordered Lists


```
<ol type="a">
      <li>List Item 1</li>
      <li>List Item 2</li>
      <li>List Item 3</li>
</ol>
```



- Numeric is default type
  - Other types can be specified
    - "1", "A", "a", "I", "i"

HTML Links (Hyperlinks)

- HTML Links created with **Link** Tags
  - `<a> </a>`
  - *`<a href="http://www.randomsite.com/">Visit Random Site.com</a>`*
- Hypertext Reference attribute gives the URL
  - `href="http://www.randomsite.com/"`
- Target attribute defines where the link opens
  - `target="_blank"` *Opens link in new window*
  - `target="_self"` *Opens link in same window*
  - `target="mywindow"` Opens in a named (possibly new) window

Exercise 3

1. Re-create the below simple FAQ for a fictional course
2. Use a heading 2 tag for the questions.
3. Use an ordered list for each question
4. URL for the Q1 link is https://www.universityofgalway.ie/science-engineering/school-of-computer-science/currentstudents/timetables/
5. URL for the Q2 link is https://www.universityofgalway.ie/science-engineering/school-of-computer-science/people/

1. **Where are the timetables for the course?**

   The course timetable can be found at the following link

2. **What are the contact details for academic faculty?**

   The programme director's info can be found at this location

Images in HTML

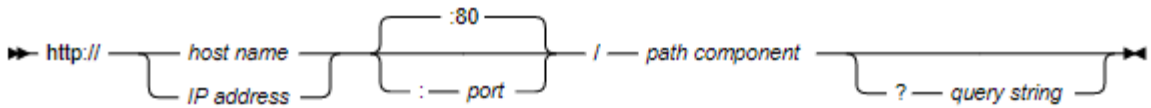
- Images are added with the empty tag <img>
- <img> requires a **source (src) attribute** with the URL of the image
  - <img src="http://www.randomsite.com/image.jpg">
- Image (and other) filenames are case sensitive on Linux and Mac servers
  - .jpg != .JPG
  - .png != .PNG

Absolute and Relative References

- URL:The path to a resource
- Any URL path in HTML can be given as either via an Absolute or a Relative Reference
- Both types are valid, interchangeable and useful in different situations
  - e.g. “a” tags and “img” tags can use both Reference Types

Structure of a URL (Absolute)

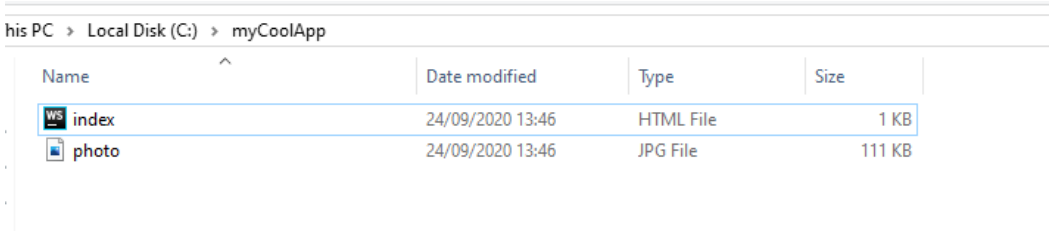
Figure 1. Syntax of an HTTP URL



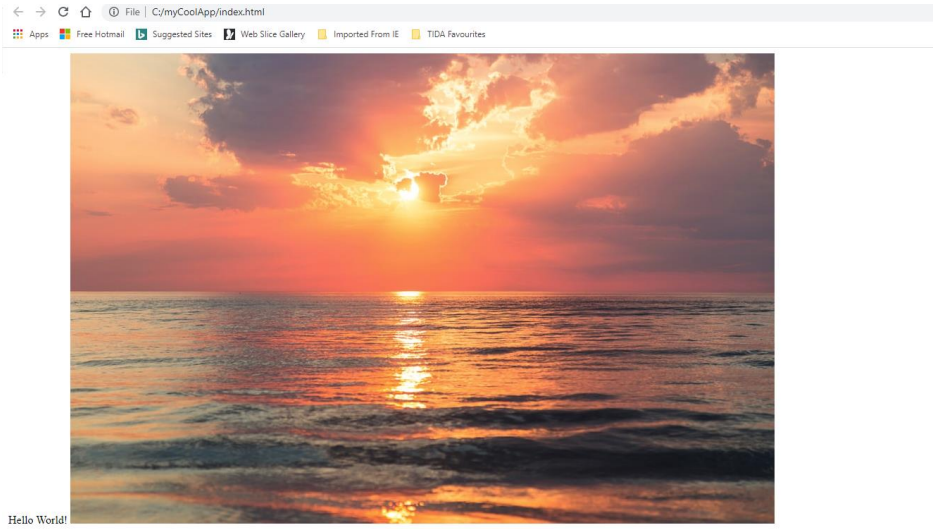
- http://www.example.com/software/index.html
- http://www.example.com:1030/software/index.html

Relative Reference

- If you wish to reference an image in the same folder you can do it via relative referencing




```
<!DOCTYPE html>
<html>
<head>
  <title>Simple
Page</title>
</head>
<body>
Hello World!
<img
src="photo.jpg"></img>
</body>
</html>
```



Exercise 4

1. Add an extra question to your FAQ page with an image of the CS building.
2. When you click on the image it opens a new tab to the CS homepage https://cs.universityofgalway.ie/.
3. Add a picture of the CS building, by including an absolute reference to an image already hosted online.
4. Also include an internal image of the building but this time relatively src it.

Final page should look something like this

1. **Where are the timetables for the course?**

   The course timetable can be found at the following link

2. **What are the contact details for academic faculty?**

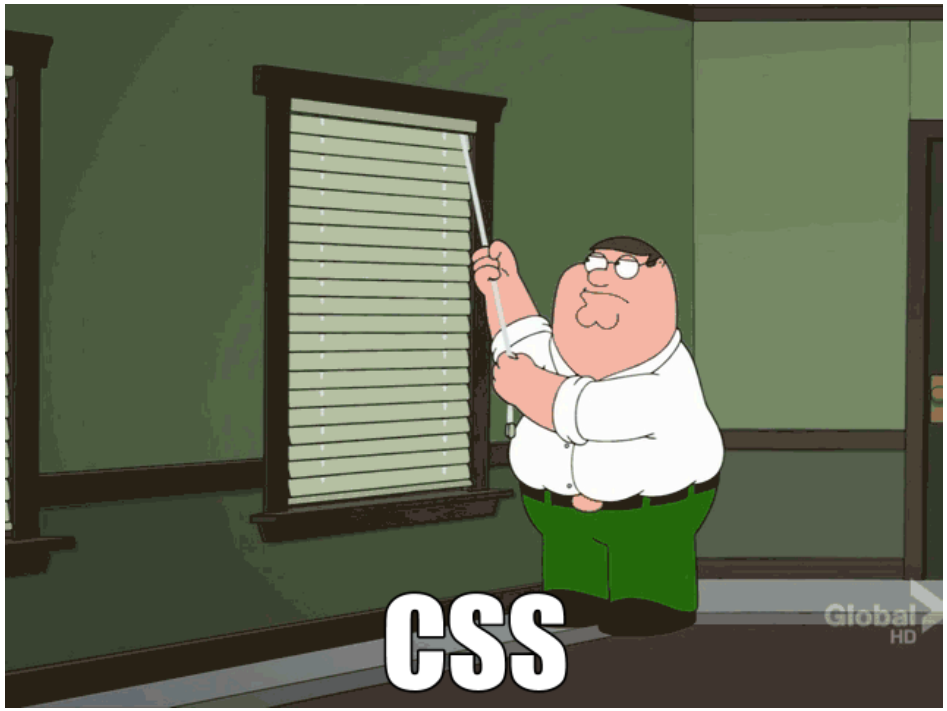
   The programme director's info can be found at this location

3. **What does the CS building look like?**

CASCADING STYLE SHEETS

What is CSS?

- Cascading Style Sheets
- Contains the rules for the presentation of HTML

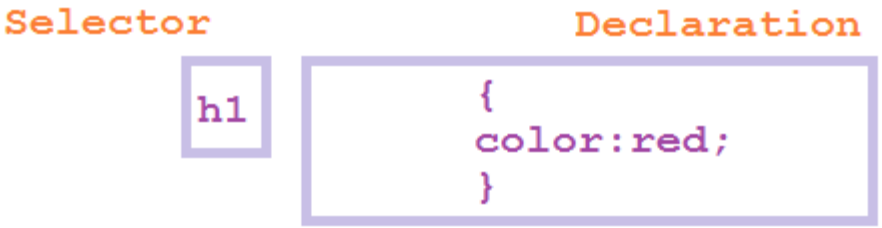


CSS Continued

- CSS allows the developer to define a set of styles
- These styles can be used across multiple pages
- Helps to create a uniform and consistent look across a web application
- Makes the formatting the content much easier
- Allows for more advanced techniques using div tags

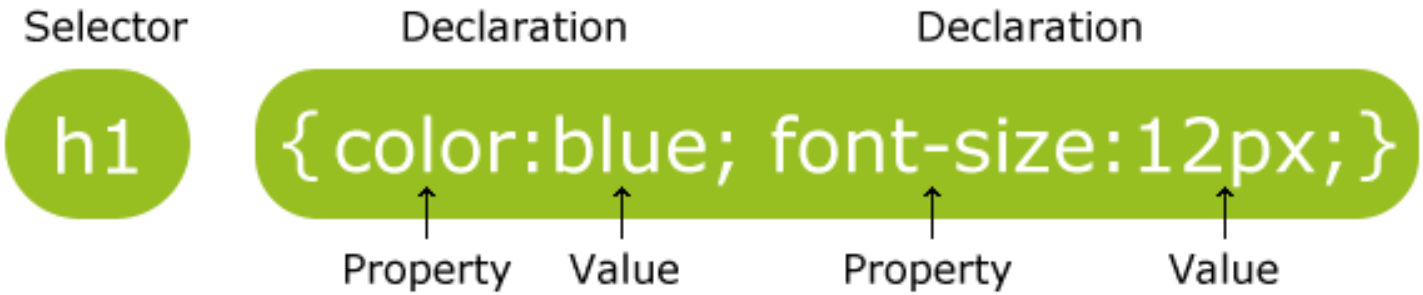
How does it work?

- CSS works by allowing you to associate rules to the HTML tags
- These rules govern how these elements should be rendered
- A rule is defined by a selector followed by a declaration block



Syntax continued

- Example of styling a Heading 1 tag
  - Colour it to blue
  - Set the text size to 12 pixels



Syntax continued


```
h1
{
 color: blue;
 font-size: 12px;
}
```

Note:

`color` **not** `colour`

Braces { } separate the selector from the declarations

Each declaration ends in semi-colon ;

Syntax continued


```
h1
{
color: blue;
font-size:
12px;
}
```

Note:

- **No space** between value and units (12 and px)
- **No quotes** for values

Sources of styles

- Inline styles
  - `<h1 style="color:red">Hello World</h1>` Hello World
- Embedded style tags
  -
```
<style>
        h1 {color:blue}
</style>
```
Hello World
- Linked styles
  - `<link href="/stylesheets/main.css" rel="stylesheet">`

Be careful with the styles

- Precedence or order over the styling
  - Browser default style (weakest)
  - User style sheet
  - Author stylesheet
  - Author embedded styles
  - Author inline styles (strongest)

- !important

Selecting a tag or element to style

- There are two main ways of doing this, class based selector and ID based

  - ID based (#)

```
<div id="content">
        My Text Content
 </div>
```

```
#content{
        width: 200px;
        }
```

  - Class based (.)

```
<div class="content">
        My Text Content
 </div>
```

```
.content{
        width: 200px;
        }
```

Font related CSS

- https://www.w3schools.com/cssref/pr_font_font.asp

Definition and Usage

The `font` property is a shorthand property for:

- font-style
- font-variant
- font-weight
- font-size/line-height
- font-family

```
<p class="p1">
        My paragraph of text
</p>
```

```
.p1 {
  font-family: "Times New Roman";
  font-weight: "600";
}
```

W3Schools have great docs on this

Exercise 1 – Adding styles

- Step 1: Add a file to the public folder of "MySampleApp" called styles.css
  - Add the following CSS to the stylesheet *styles.css*
    - A class based selector (".") called "links"
    - Set the font size to 16px
    - Set the font colour to blue
  - Add this linked stylesheet to your page (see slide 4)
- Step 2: Add 3 random hyperlinks to the web page
  - Associate this new class (created in Step 1) with all links on the page, does it work?
- Step 3: Using inline styling alter the colour of one of the links to change the font colour to green. Does it override the CSS in Step2?

The CSS Box model

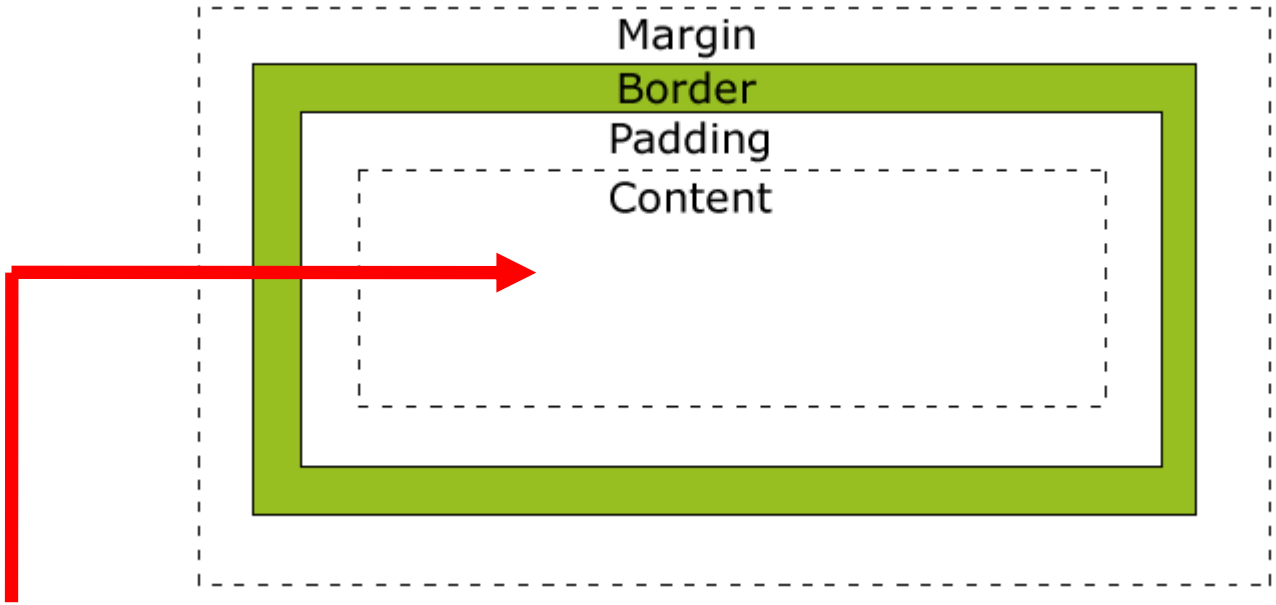
- All HTML elements are considered as boxes
  - Paragraphs, Headings, Tables etc.
- Box Model
  - A box that wraps around all the HTML elements
- Consists of Four Parts
  - Content
  - Padding
  - Border
  - Margin

CSS Box Model

- Content
- Padding
- Border
- Margin

- Padding, Border & Margin are ZERO by default

Content Area



- Content
  - The HTML element concerned
  - Text, Image, List, Table, etc

Content – Embedded <p> Example

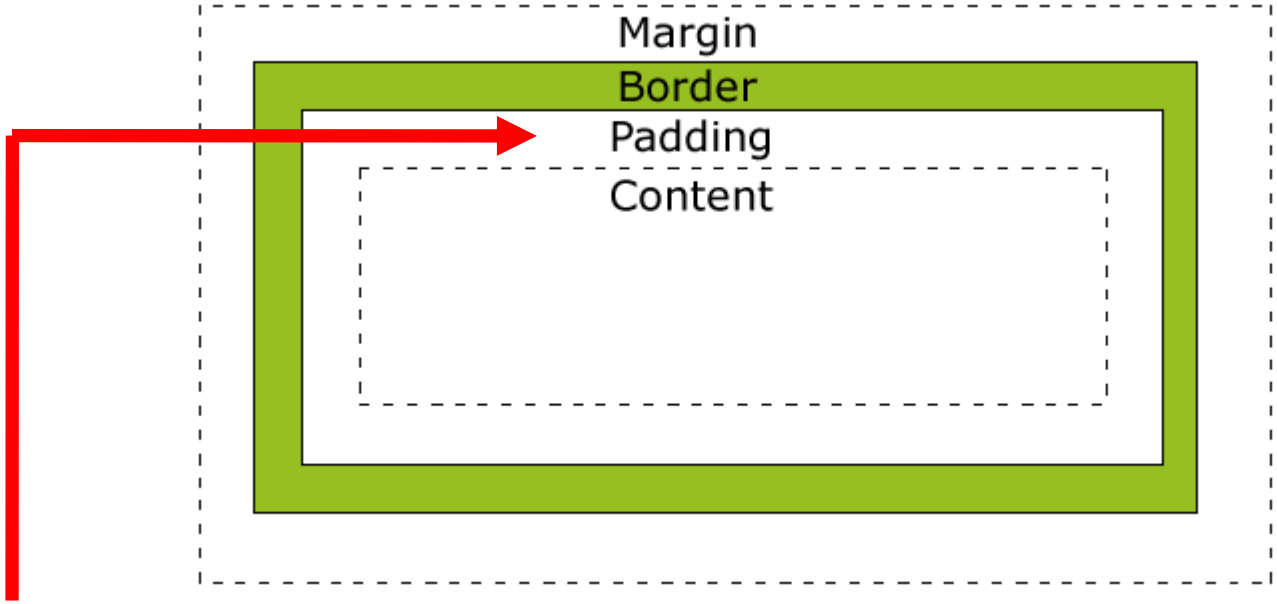

```
<html>
<head>
<style>
p
{
  background: #00FFFF;
}
</style>
</head>
```

```
<body>
<p>Text Goes Here!</p>
</body>
</html>
```

Result:

Text Goes Here!

Padding Area



- Padding
  - Empty space surrounding the Content
  - Uses the **same** background colour as the Content

Padding Content

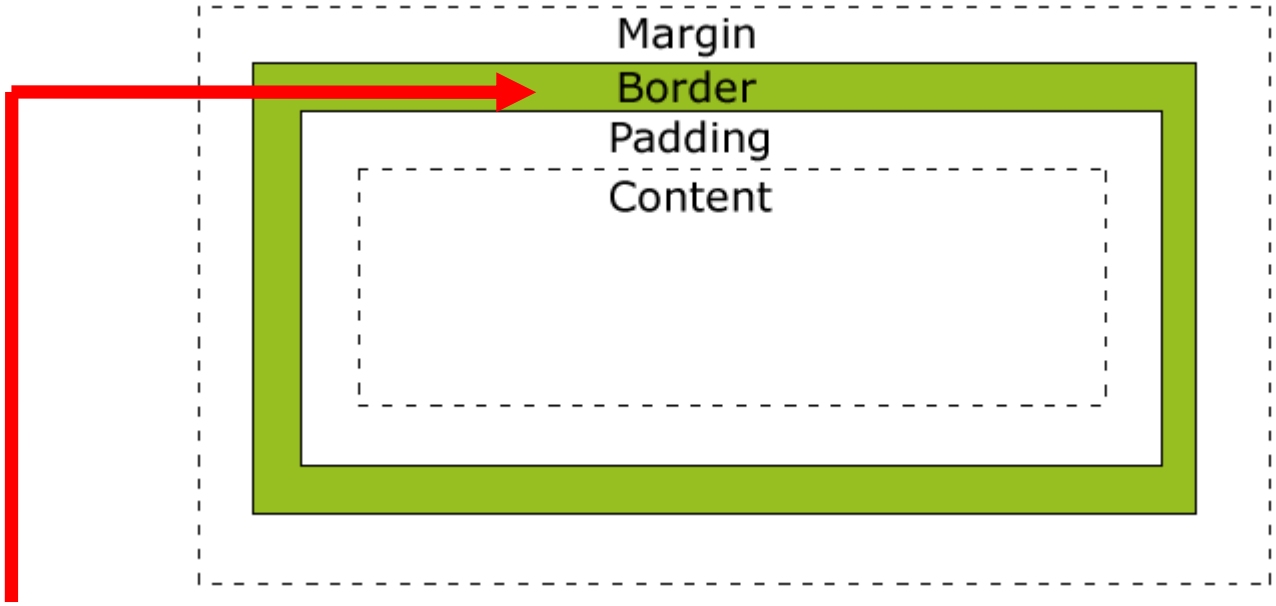

```
p
{
background: #00FFFF;
padding: 0px;
}
```

Text Goes Here!

```
p
{
background: #00FFFF;
padding: 20px;
}
```

Text Goes Here!

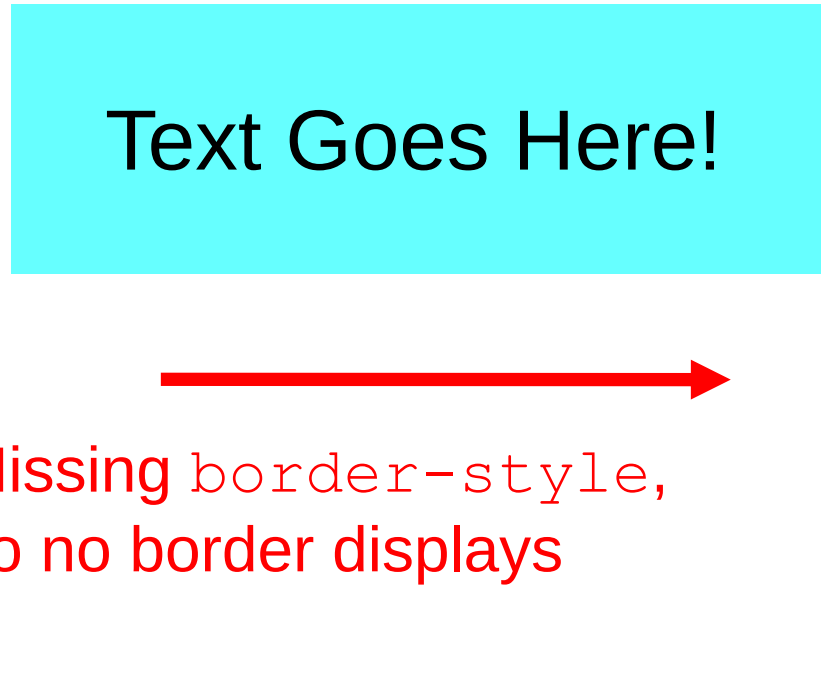
Border Area



- **Border**
  - The HTML element concerned
  - *border-style* **must** be set for border to take effect

Bordering Content


```
p
{
background: #00FFFF;
padding: 20px;
border: 20px;
border-color: #FF0000;

}
```

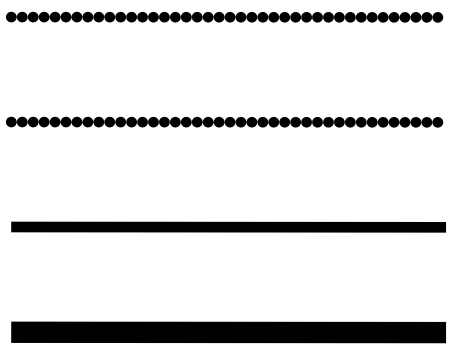


Bordering Content


```
p
{
background: #00FFFF;
padding: 20px;
border: 20px;
border-color:
#FF0000;
border-style: solid;
}
```

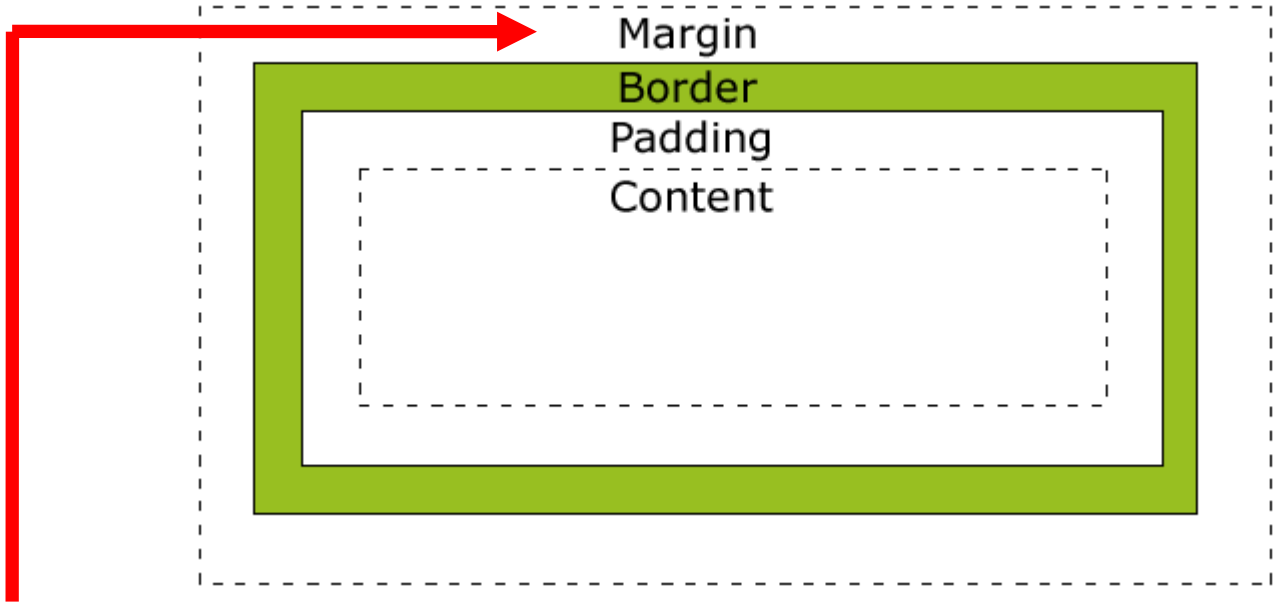
Border-style values

- none — *No border*
- dotted — *Dotted border*
- dashed — *Dashed border*
- solid — *Solid border*
- double — *Two solid borders*



- groove — *3D "grooved" border (engraved)*
- ridge — *3D "ridged" border (emboss)*
- inset — *3D "inset" border (lowered)*
- outset — *3D "outset" border (raised)*

Margin Area



- Margin
  - Transparent area that surrounds everything else
  - Used for spacing the element relative to others

Exercise 2

- Create a paragraph of text “Hello World”
- Add a background colour of your choosing using Hex values
- Add padding of 10px all around it
- Add a 1px dashed border around it of a colour of your choosing.
- Place a second paragraph below the first and style it the same way as the first however place a 10px margin between the two (either bottom of the first or top of the second).

Grouping and descendants

- Multiple selectors can be grouped in a single style declaration

```
p, .main{
        font-weight: bold;
}
```

- Select elements that are descendants

```
<div class="abc">
 <div>
  <p>
     My Text Content!
  </p>
 </div>
</div>
```

```
div.abc p{
        font-weight: bold;
}
```

CSS values

- *Text-align:center;*
- Numerical values: Numerical values are usually followed by a unit type.

  font-size:12px;
- 12 is the numerical value and px is the unit type in pixels
  - Absolute values :in, pc, px, cm, mm, pt
  - Relative values: em, ex, %
- Color values: color:#336699
  - Blue, red, green etc…

Colour Wheel [visible light]

- Combining different levels of RGB yields different colours
- Wheel on the right shows colour relationships
- Yellow is made of equal parts Re
  - Thus, HTML Colour for Yellow:
  - #FFFF00

The div tag

- Defines a division of a HTML page and often used as a container for other elements.
- Block level elements such as div's, paragraphs headings sit on top of each other, by default.

HTML

```
<body>
        <div id="div1"></div>
        <div id="div2"></div>
        <div id="div3"></div>
</body>
```

CSS

```
#div1 { width:300px;background:yellow;}
#div1 { width:300px;background:blue;}
#div1 { width:300px;background:orange;}
```

Inline elements

- Inline elements such as span and img sit side by side

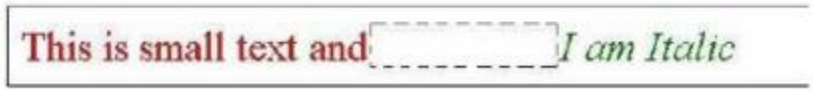

```
<div id="row1" >
   <span class="norm">This is
small text and </span>
   <span class="big">this is big</
span>
   <span class="italicText"> I am
Italic</span>
</div>
```

```
.norm {
        color:red;
}
.big {
        color:blue;
        font-weight:bold;
}
.italicText {
        color:green;
        font-style:italic;
}
#row1 {
        padding:10px;
        border:solid 1px #000;
}
```

Visibility

Visible : The element is visible (default).
Hidden : The element is invisible (but still takes up space)


```
.big {
        visibility:hidden;
}
```



Exercise 3 – Page styling

- Create two paragraphs of text
  - This is a paragraph of text
  - This is a paragraph of text
- Surround both paragraphs with a single div tag
- Create a style (using class-based selection) which centres the text of the paragraphs and changes the colour to #663399. Place the style in a separate styles.css file. Apply it to the div element.
- Within each paragraph use a span to change the word “paragraph” to yellow. Add the style to the styles.css file and using class-based selection apply it to both spans.

BOOTSTRAP

https://getbootstrap.com/

What is Bootstrap?

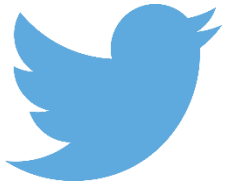
- Bootstrap is a CSS framework, it is CSS classes for structure, layout, components (buttons, navbar, modal etc), forms, written by other developers.
- This enforces a uniform layout, look and feel on the web application.
- Freely available to develop web sites and web applications.

What is Bootstrap cont.

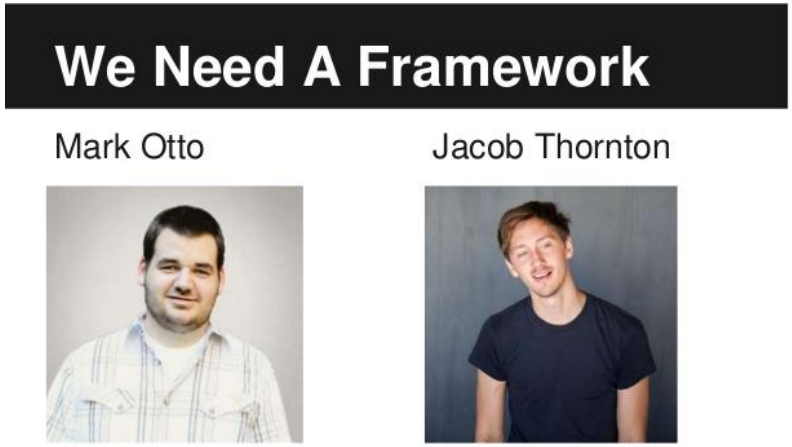
- JavaScript is also used in conjunction with the CSS classes, for things like animations, transitions, popups etc.
- The CSS within it is quite detailed, there are lots of classes with varying levels of hierarchy.
- It's fully customisable and the web is full of themes and templates for apps built using it
  - https://themes.getbootstrap.com/

Who developed it?

- It was developed by Twitter's Mark Otto and Jacob Thornton



- They wanted to standardise the frontend toolsets across twitter
- It was released as open source in August 2011 on Github



Adding Bootstrap to our web apps

- Two options
  - Download the files, i.e. CSS, JS, place them in the app folders
  - Use a Content Delivery Network URL

- https://getbootstrap.com/

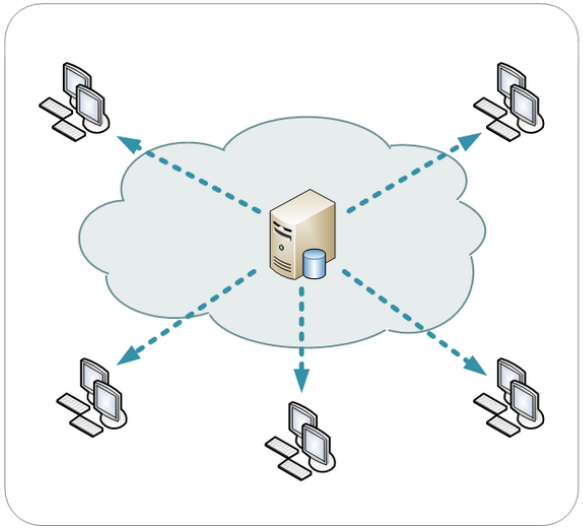
Content Delivery Networks

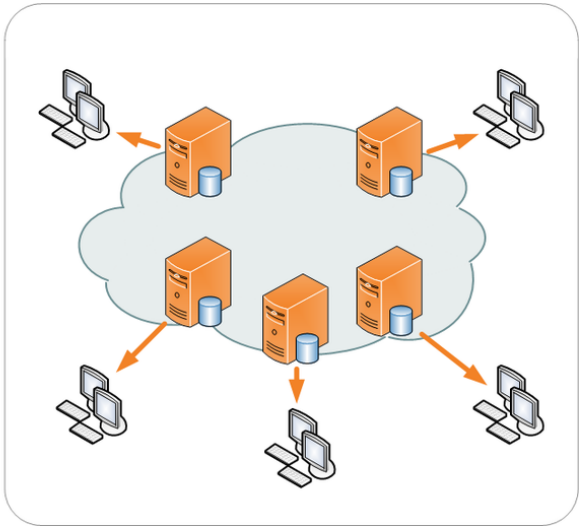
- A popular way to include frontend libraries and technologies is to use a CDN
- Instead of having a single server, CDNs involve using a collection of servers to serve content
- These servers are usually placed geographically close the user base to ensure that maximum performance is achieved
- Largely designed for delivering static content, images, videos, and web content such as text, graphics and scripts.

CDNs are popular

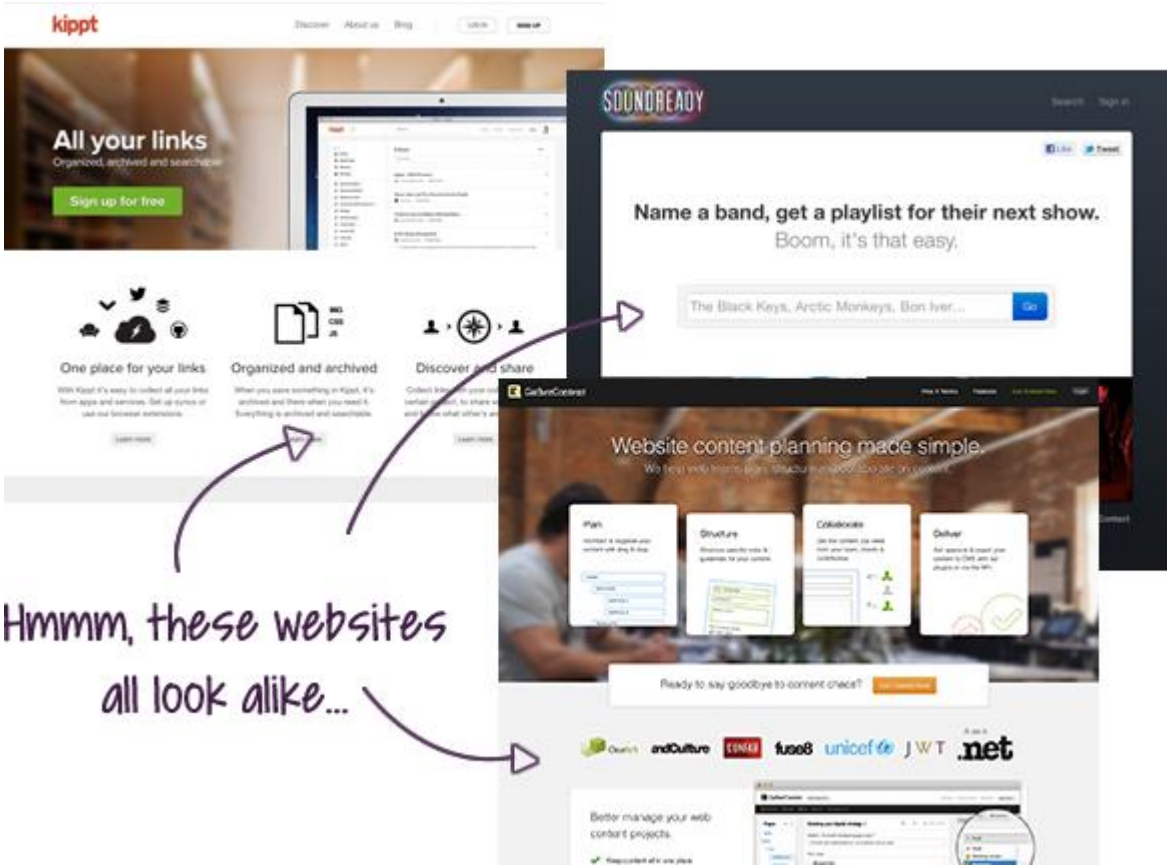
- Almost every frontend technology provider, Facebook, Google, Twitter will provide access to their libraries via a CDN

- This is very useful from a caching perspective, as browser caches should already have a hit for a regularly used CDN address, such as that from Bootstrap





How to recognise it?

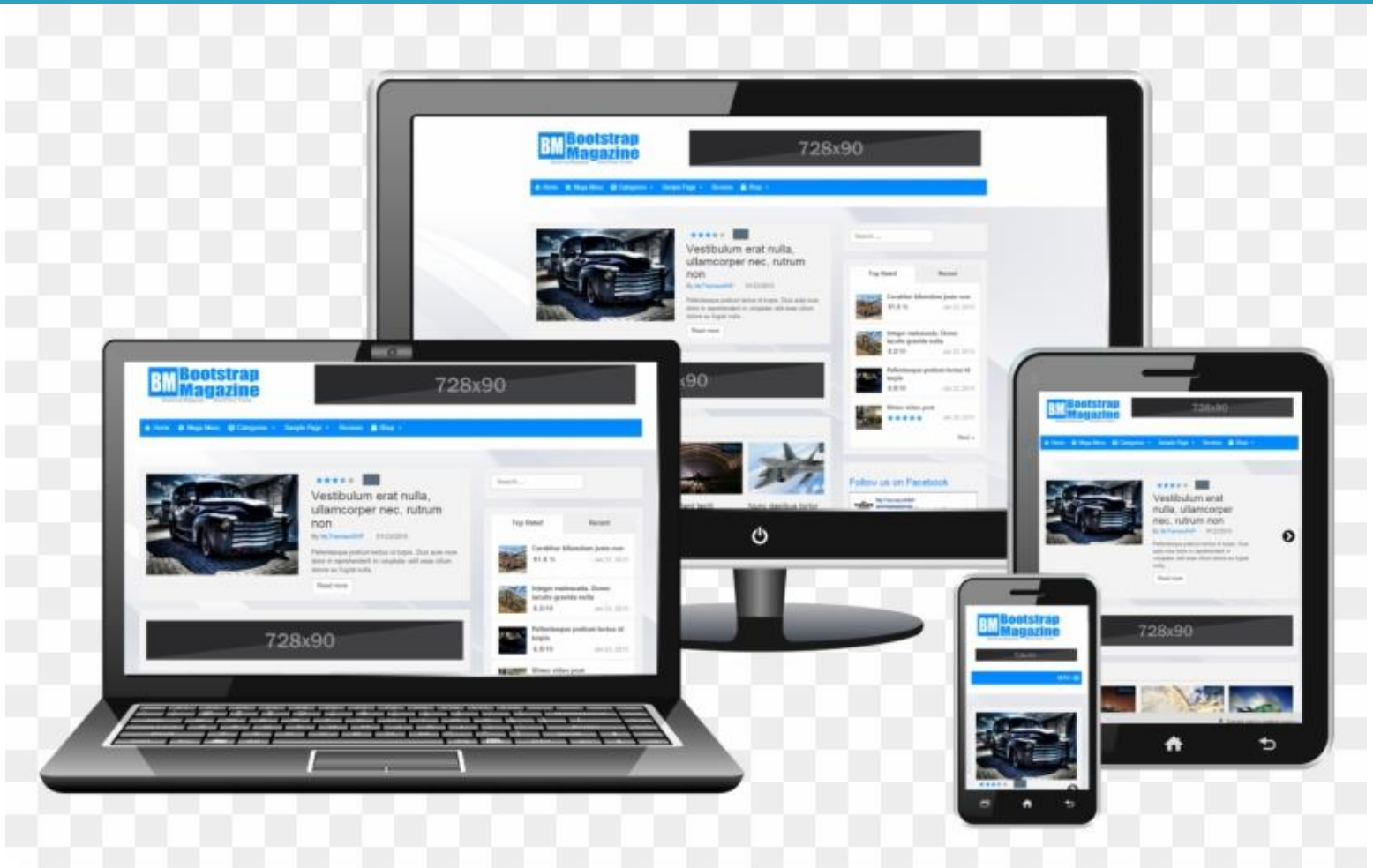


Supported by all browsers

Solid cross browser compatibility

All browsers have different
rendering engines, webkit, trident …

Responsive Design



https://www.pngfind.com/mpng/iJmbRw_bootstrap-responsive-design-laptop-tablet-mobile-psd-hd/

What is responsive layout?

- Produces an optimal viewing experience for the user independent of the device they are viewing it on
- Bootstrap is mobile first (software that has been developed to prioritise use on mobile platforms)
- If you view it on a mobile, tablet or larger screen it will scale accordingly
- As the viewport size increases it can scale up to 12 columns

https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-responsive-layout.php

Get Bootstrap

- Navigate to https://getbootstrap.com/
- Scroll down to the Include via CDN

- Copy the CSS only link and paste it in between the <head></head> tags at the top of the page
- Take the JavaScript and pop it in below it.

Add Bootstrap to our Web pages

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title> Welcome to my cool new web page </title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
    <link rel='stylesheet' href='https://cdn.jsdelivr.net/npm/bootstrap@currversion…' />
    <script src='https://cdn.jsdelivr.net/npm/bootstrap@currversion…' />
  </head>
  <body>

  </body>
</html>
```

Add the bootstrap CSS URLs and JS URLs from getbootstrap.com

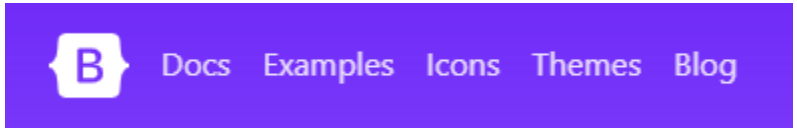
Bootstrap container class

- The container class is a fundamental building block of Bootstrap.
- They are required for the Grid system to work
- Thus it is best to ensure that all of your HTML elements marked up with Bootstrap classes are nested within a container

```
<div class="container">
…
</div>
```

Bootstrap Exercise 1 – Add bootstrap

- Add Bootstrap to your apps (see previous slide)
  - Test that it works by clicking on the Docs section (getbootstrap.com) and the choosing “Buttons” component
  - Paste the button samples into your HTML page (index.html)



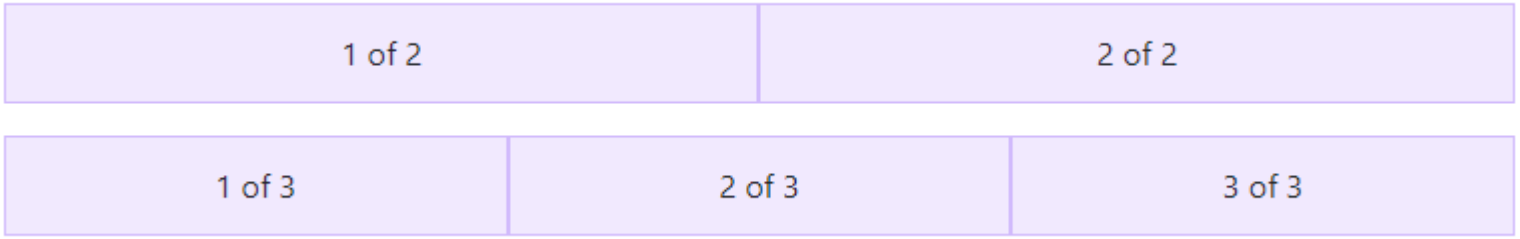
  - Does it look like the example in the docs, if not you haven’t included Bootstrap properly

  - Add a navigation bar to your app (consider where you might place this)

Bootstrap grid system

- For laying out content on your pages Bootstrap supports a grid system which structures the page into rows and columns

https://getbootstrap.com/



- It supports a concept of Rows and Columns, like a spreadsheet.
- Rows contain columns, columns contain the content!
- The total number of columns is 12.

Bootstrap grid system

- It uses div tags and with specific classes associated with each div.
- The rows and columns are all div tags
- It is built with Flexbox (https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox)

How the “Grid” system works

- It supports six responsive breakpoints, which allows it to adjust the views for different screen sizes, **small, medium, large** etc.
- It has predefined padding and horizontal gutter widths between columns which are customizable but keep the content nicely spaced and uniformly structured with each other.

How the grid system works

- Rows must be placed within a **.container** class for proper alignment and padding.
- Use rows to create horizontal groups of columns.
- Content should be placed within columns, and only columns may be immediate children of rows.
- Predefined grid classes like **.row** and **.col** are available for quickly making grid layouts.
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on **.rows.**
- Grid columns are created by specifying the number of twelve available columns you wish to span. For example, three equal columns would use three **.col-4** divs.

Bootstrap column classes

	xs <576px	sm ≥576px	md ≥768px	lg ≥992px	xl ≥1200px	xxl ≥1400px
Container `max-width`	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`	`.col-xxl-`
# of columns	12					
Gutter width	1.5rem (.75rem on left and right)					
Custom gutters	Yes					
Nestable	Yes					
Column ordering	Yes					

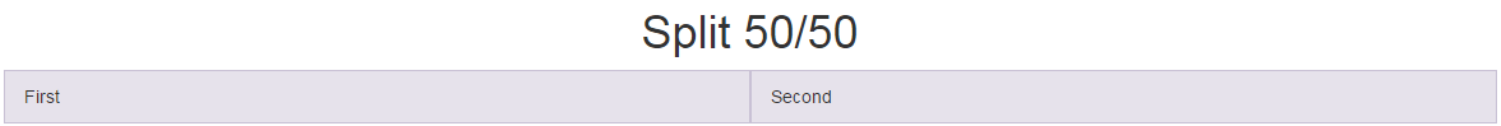
https://getbootstrap.com/docs/

Example

- Lets say you want to split the screen 50/50 for large screen devices

```
<div class="container">
    <h1>Split 50/50</h1>
    <div class="row show-grid">
        <div class="col-lg-6">First</div>
        <div class="col-lg-6">Second</div>
    </div>
</div>
```

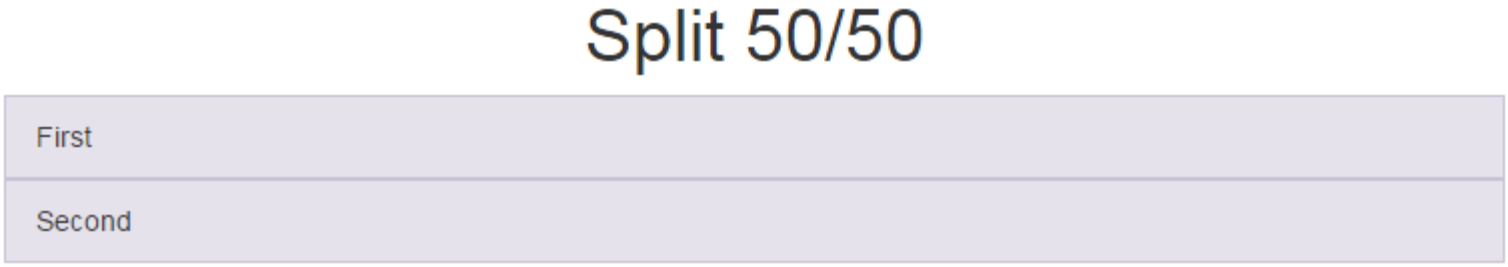
Remember everything is 12 columns wide
Note that this will affect breakpoints above it also



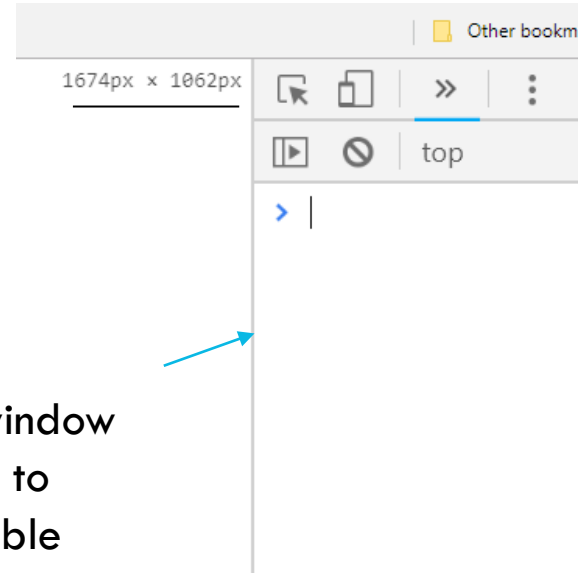
- On large screen sizes the columns will be placed side by side

Resize the screen

- When the screen is resized i.e. less than 992px it will stack the two columns



- You can find out the pixels by enabling the Chrome debugger



Keep them 50/50 on all devices

- If you want to keep them 50/50 on all devices you can specify the lowest size and it will scale

Practice it now!

Exercise 2 - Bootstrap – Grid

Provide the HTML and corresponding Bootstrap classes to split a row into two separate sections, each sized 6 columns wide on large and medium devices or above. In the case of smaller devices, the columns should stack on top of each other.

JAVASCRIPT

Conditionals and Loops

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Boolean Variables and Expressions

- Boolean literal may be either **true** or **false**
  - **let** isLoggedIn = true;
  - Note that true is a **value** not a string
  - *Do not enclose in quotes!*
  - Boolean expression is one which resolves to either true or false

Conditional Statements

- **if** statement – execute code only if some condition is true
- **if…else** statement – execute some code if the statement is true and another piece of it is false
- **if…else if… else** statements – used to execute one of many blocks of code

Conditional Statements JS


```
// If statement
if(true)
{

}

//If else statement
if(true)
{

}
else{

}
```

```
// If else if else
if(true)
{

}
else if(false){

}
else{

}
```

Relational operators

- Less than <
- Greater than >
- Less than or equal to <=
- Greater than or equal to >=
- Equal to ==
- Not equal to !=

Exercise 4: Conditional Example

- Write code that checks whether a given salary is well paid or poorly paid using conditionals.
  - Create a variable called *salary* and set it to 5000
  - Create an if statement which checks if *salary* is greater than 100000 and alerts "Wealthy Salary"
  - If less than 10000 alert "Poor Salary"
- What happens if you input 30000?

Logical Operators

- AND (&&)
  - Compares two Boolean values and equates to true if they are both true
- OR (||)
  - Compares two Boolean values and equates to true if one or the other (or both) is true
- NOT(!)
  - Negates the value of a Boolean value

Exercise 5: Conditional Example

- Extend exercise 4 to check if the salary is within 100,000 and 10,000 if so then alert “Normal Salary”

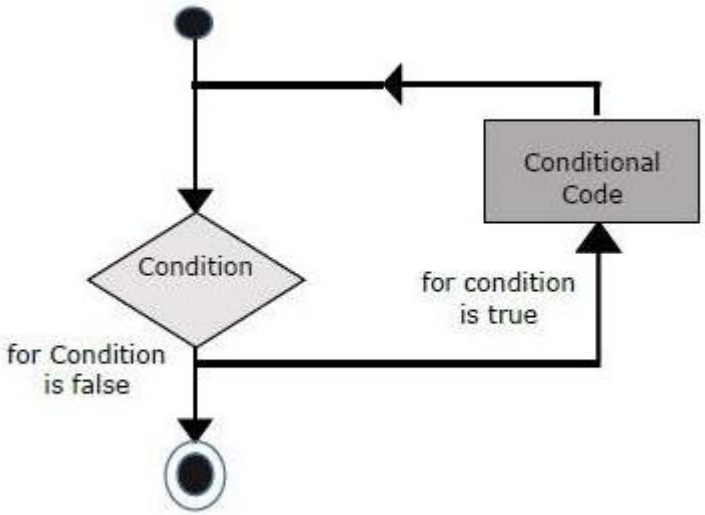
Loops in JavaScript

- Repeat a block of code
  - While some condition holds true
- For loops
  - Execute code a specific number of times
- While loops
  - Execute code an undetermined number of times

For Loop

- The For Loop repeats a block of code a certain number of times

```
//Repeat while i is still
less than 10
for(let i=0; i<10; i++)
{
    // Execute code
    // i++ at the end
}
```



https://www.tutorialspoint.com/javascript/javascript_for_loop.htm

While Loop

- Executes code an undetermined number of times
- Loops while the condition remains true
- Condition must be true for code to execute

```
while(i<j)
{
    // Execute this code
}
```

Break and Continue

- break;
  - Exits a loop immediately when it is encountered
- continue;
  - Stops the current execution of a loop
  - Does not exit the loop
  - Goes to the top of the loop for the next iteration
- Both can be used on any loop construct

Exercise 6: Loops

- Create a loop, either for or while, that alerts the even numbers between 1 and 10;

INTRODUCTION TO JAVASCRIPT

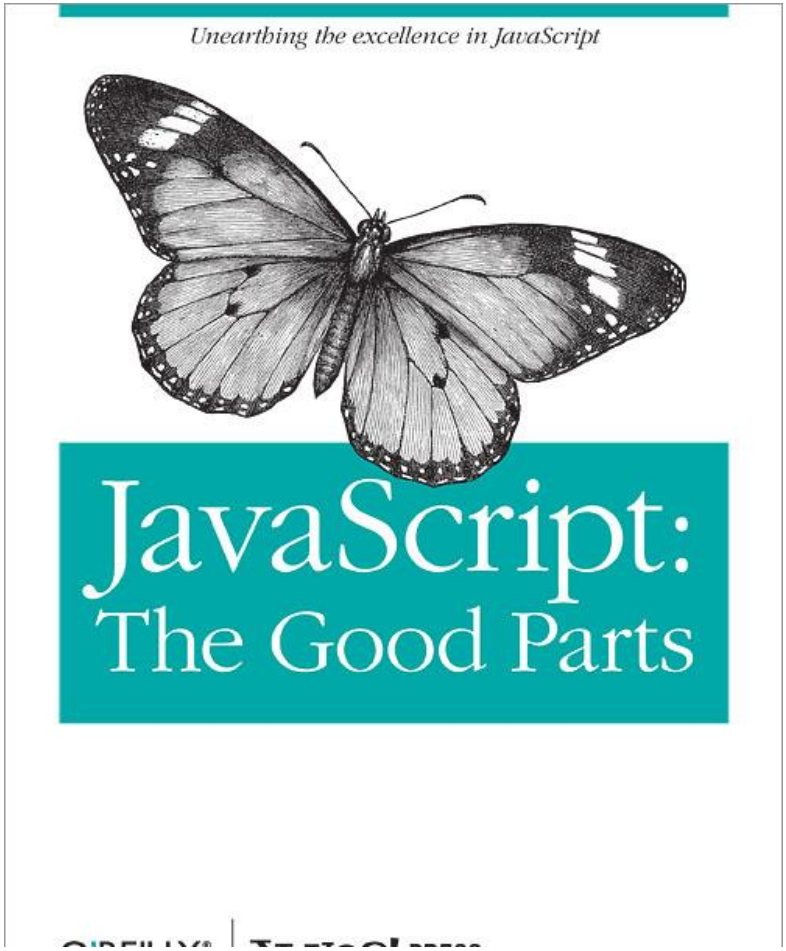
Client-side JS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

JavaScript

- Best text book for learning JS

- Douglas Crockford

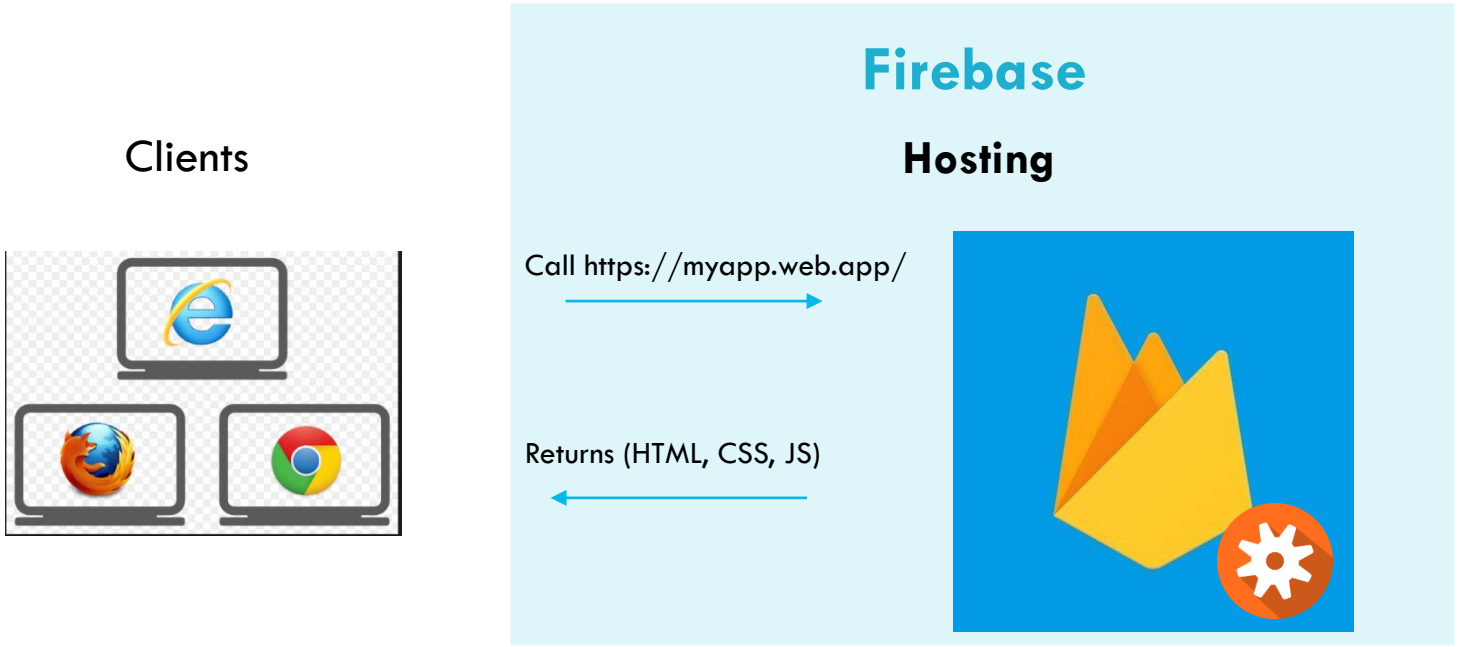


Overview

- Introduction to JavaScript
  - Adding client side JavaScript to our web pages
- JavaScript fundamentals
  - Variables
  - Types
  - Numbers
  - Strings
  - Booleans

Client side JS

- Client side JS
  - JS code that runs **locally** on the users machine/device (in the browser)



Client side JS cont.

- It allows you to manipulate/update the HTML content based on the users actions
  - If the user clicks a button “read more”, you can expand the content on the page. – News Blog
  - If the user continues scrolling further down, you can send a request to the server for more data. – Twitter feed
  - Sections of the page can be updated without reloading the entire page. – Dynamic Dashboards
- Brings an interactivity to what are essentially static HTML pages

JavaScript Syntax

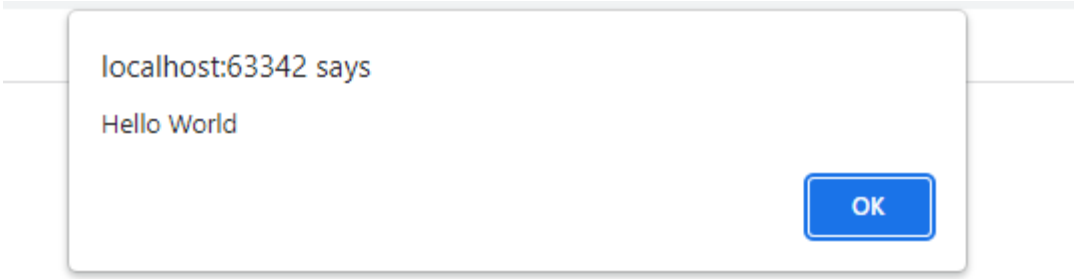
- 'C like' language
- Braces used to denote code blocks {} – like Java
- Semi-colons used at the end of lines
  - If you leave them out it will still compile, semi-colon insertion is automatically done during parsing so in most cases your code won't break if you leave them out.
- Comments as per C
  - Single line comments // Single line
  - Block comments /* Block comments
    can span multiple lines */
- Comments in HTML
  - <!-- This is my comment -->

JavaScript on HTML pages

- Similar to CSS we can include client-side JavaScript on HTML pages via the script tags

Inline JavaScript

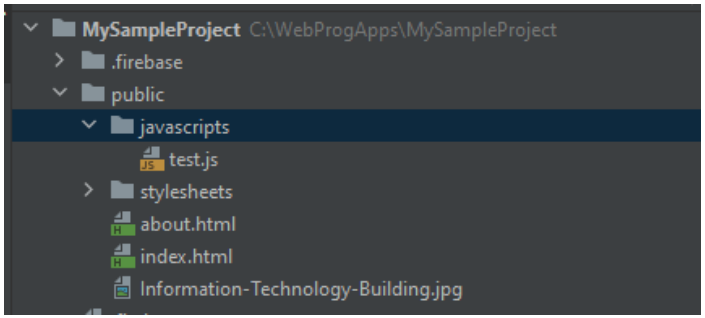
```
<script>
    window.alert("Hello World");
</script>
```



Inline vs External

- As with CSS where we can embed the CSS within the page or define a separate stylesheet, JavaScript also supports this.
- You can create **external JavaScript files** and using the **src** attribute on the script tag reference the URL

```
<script
src="javascripts/test.js"></script>
```



The Window Interface

- A Window Interface represents a window containing a HTML document.

- A window object is exposed to your JavaScript code and you can call various methods
  - window.alert("string") //displays a popup alert box
  - window.prompt("string") // displays a prompt where values can be entered by the user
  - window.confirm("string") // Confirmation box, allowing one to figure out what the user has pressed

Activity 1: Add some JS to our apps

- Create a new JS file. Pop it into the JavaScripts folder in your app (you will have to create this folder too)
- Write the following line in the JS file alert("Hello World");
- Save the file as hello.js
- Include the script on the page "hello.js" to the folder <mySampleApp>/public/javascripts

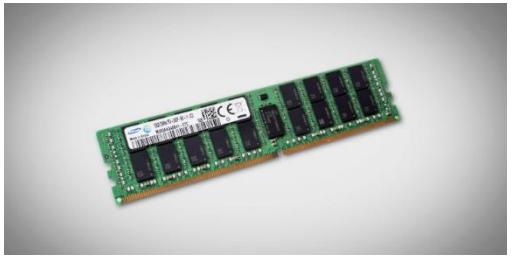
JAVASCRIPT

Variables - Numbers, Strings and Booleans

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Variables (Numbers)

- Variables are the names you give to computer memory locations which contain data in your computer programs
- Created using the **let** keyword (Let was introduced in ES6 2015)
  - **let** age = 24;
- Variables can also be created using the **var** key word but this essentially makes them globally scoped which can be problematic.
  - **var** age = 24;
- If you don't want the variable to change then use **const**
  - **const** age = 24;
- Unlike languages such as C, variables do not need to be declared before being assigned a value
  - **let** age = 24;
  - Variable "age" will be created if it isn't already
- Declare multiple variables
  - **let** age1, age2, age3;



Arithmetic in JavaScript (Numbers)

- Similar to C / C++ / Java etc.
- Basic **Arithmetic Operators**

Addition	+	ans=a+7
Subtraction	-	ans=a-b
Multiplication	*	ans=a*b
Division	/	ans=a/b
Modulus	%	ans=a%b

Exercise 2: Working with numbers

- Declare a variable called *salary* and assign it a value of 40000

- Assume that the €40k is your salary and you've just been awarded a bonus of €1000, so add this to the *salary* and using an alert display it on the page.

Variables (Strings)

- A **string** is a group of characters
- A **string literal** is a group of characters enclosed in quotes
  - “This is a string literal”
  - “This is too”
  - This isn’t
- A **string variable** is a variable that holds a string
- A **String** is a data type in JavaScript
- Space is a valid character in a string

String operators

- JavaScript supports string operators to join two strings together
- **let** name = “Enda ” + “Barrett”

- Can also concatenate string variables

```
let first, last, full;
first = “Enda ”;
last = “Barrett”;
full = first + last;
```

Variable types

- Variables in JavaScript are not associated with any particular type and any variable can be assigned (and re-assigned) values of all types.
- JavaScript supports dynamic or weak typing
  - This means that it will resolve the appropriate type at compile time, based on the input values
- **This does not mean that types don't exist, they do.**
  - Number.
  - BigInt
  - String.
  - Boolean.
  - Null.
  - Undefined.
  - Symbol.

```
let foo = 42; // foo is now a number
foo = "bar"; // foo is now a string
foo = true; // foo is now a boolean
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

Exercise 3: Add string and number

- Create a variable *salary* and assign a value of 40000
- Create another variable *bonus* but assign it a value of "1000"
- Create a third variable called total which adds *salary* and *bonus* together
- Try adding 1000 euro bonus to the salary, what do you notice?

Adding a string to a number

- When you try to add a string and a number together the JavaScript JIT compiler will try to complete automatic type conversion. In this instance it converts the *number* 40000 to a *string* and concatenates it to the *string* bonus of “1000”.

Boolean Variables and Expressions

- Boolean variables are a special type of variable which can have two specific states **true** or **false**

- Declaring a Boolean variable
  - **let** isVar = true;
  - Note that **true** is a **value** not a string
  - *Do not enclose in quotes!*
  - Often pre-fixed with the naming convention "is"

Booleans

- Booleans are typically used in expressions, to evaluation one thing against another.

- If you wanted to check whether one number was larger than another number, you would place the two numbers within an expression and the result would be either **true** or **false.**

- Which leads to conditionals!

Variable Naming Rules

- Variable **names** known as **identifiers**
- **Identifiers** are case sensitive
  - myAge
  - MyAge
  - myAGe
  - All different variables above
- Must begin with a letter or underscore
- Cannot contain spaces
- Cannot use keywords

```
let age = 20;
```

JAVASCRIPT

Functions

Functions in JavaScript

- A JavaScript function is a block of code designed to perform a particular task.
- The function is executed when "something" invokes it (calls it)
- f(x) = x + 2

```
function multiply(param1, param2)
{
        return param1 * param2
}
```

Function won't execute unless it is invoked

Functions in JavaScript

- A function can be named or it can be anonymous

```
function(param1, param2)
{
        return param1 * param2
}
```

- The params are items of data that the function needs to perform its task
- Not passing a required parameter will result in an error
- Can have zero parameters but still requires empty parameters

ES6 Arrow functions

ES6

```
(param1, param2) =>
{
        return param1 * param2;
}
```

ES5

```
function(param1, param2)
{
        return param1 * param2;
}
```

- Arrow functions are more concise, developer can achieve the same functionality with fewer lines of code.
- Support concise function expressions

Assign function to a variable


```
let multiply =  (param1, param2) =>
{
   return param1 * param2;
}
multiply(3,3)
```

```
let multiply = function(param1, param2)
{
   return param1 * param2;
}
multiply(3,3)
```

Returning from a function

- Functions can return values to their calling environments

- Use the ***return*** statement to do this

```
function divide(numerator, denominator)
{
        return numerator / denominator
}
```

Returning from a function

- The returned value from a function can be assigned to a variable.

```
function incrementAge(myAge)
{
    myAge++;
    return myAge;
}

let incAge = incrementAge(26);
alert("Incremented age is " + incAge);
```

Invoking the function and passing arguments

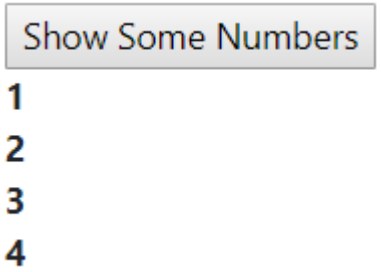
Functions

- Functions consist of
  - Unique name (cannot be keywords)
    - If they are named!
  - Parameters (again cannot be keywords)
  - **Don't** declare variables as parameters (**let** param1)
  - Code block to execute
  - Will only return once, but can use conditional statements to control the execution of the code and have multiple return statements

Invoking a function from HTML


```
<script>
function numbers() {
   let sHTML = "<b>";
   sHTML += 1 + "<br>" + 2 + "<br>" + 3 + "<br>" + 4 + "<br>";
   sHTML += "</b>";
   clientSideContent.innerHTML = sHTML;
}
</script>
<button onClick="numbers()">Show Some Numbers</button><br>
<div id="clientSideContent">Something here</div>
```

DOM Manipulation



Inline JS within our pages
-Note we can also link to a JS file too!
-Try and reproduce the same functionality but this time by using a separate JS file

Exercise: Functions

- Copy the JS code from the previous slide and place it into a separate JavaScript file.
- Modify the function “numbers” to accept a param
- If the argument passed in is 1 then the numbers printed out should be 1, 2, 3, 4.
- If the argument passed in is something else, then the numbers printed out should be 2, 4, 6, 8

JAVASCRIPT

Events

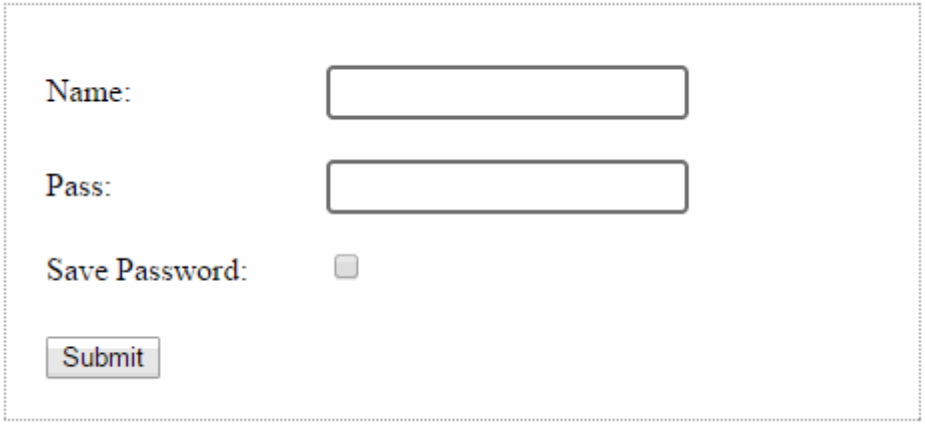
OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Events

- Actions that can be responded to by JavaScript
- Every element on a page has certain events which can trigger some JavaScript code
  - We can identify when a user clicks a button with the **onClick** event
  - Can then assign a function to run when the event is identified
  - Events defined as an attribute in the **HTML Tag**

Examples of Events

- A mouse click
- A web page or an image loading
- Moving the mouse over a hot spot on the web page



Other Mouse Events

- **onClick**
  - Triggered when the mouse clicks an element
- **onMouseDown**
  - Triggered when the mouse button is pressed
- **onMouseUp**
  - Triggered when the mouse button is released.

Selecting and De-Selecting Elements

- All three normally used with form input elements (text boxes, buttons etc.)
- onFocus
  - Triggered when an element gets focus
  - E.g. an element that is clicked is said to be in focus
- onBlur
  - Triggered when an element loses focus
- onChange
  - Triggered when the content of an element changes

As the mouse moves over HTML elements

- **onMouseOver**
  - Triggered for an element when the mouse cursor is moved over that element
  - e.g. moving the mouse over an image ('rollover')
- **onMouseOut**
  - Triggered for an element when the mouse cursor is moved away from that element
  - e.g. moving the mouse out of the image

Other Keyboard Events

- onKeyDown
  - Triggered when a keyboard key is pressed
- onKeyUp
  - Triggered when a keyboard key is released
- onKeyPress
  - Triggered when a keyboard key is pressed or held
- onSelect
  - Triggered when text is selected

Exercise: Multiply numbers from text input

- Place a text input on your web page
- Ask the user to pop in a number into the textbox using the placeholder attribute
- Place a button underneath it called “Multiply”
- When the user clicks on the button a function is invoked which takes the value from the text box, multiplies it by 3 and pops it back into the same text box
- Note that if you place an ID on the input, you can access the value the user enters via myelement.value

JAVASCRIPT OBJECTS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

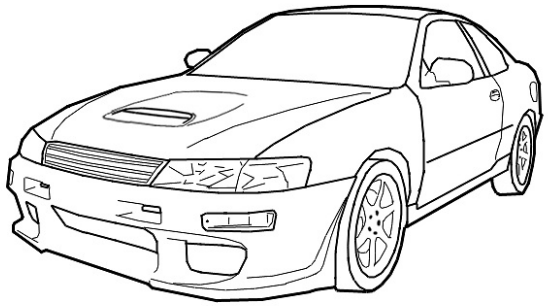
Objects

- We have seen how we can define primitives and store information using variables
- For example:
  - `let num = 123;`
  - `let str = "Enda B";`
- What if we want to represent something a little more abstract?

How would you represent these?

Sales person

Smartphone

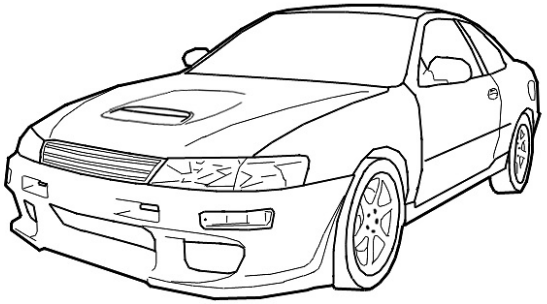


Car

Objects / User Defined Objects

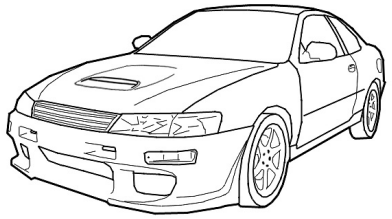
- Sales person
  - Name (String)
  - Phone number (Number)

- Car
  - Make (String)
  - Model (String)
  - Age (Number)



Object Example

- Objects are variables too, but contain many values
- **let** car = {make: "Ford", model: "Mondeo", age:2};
- The values are expressed in a *property:value* format



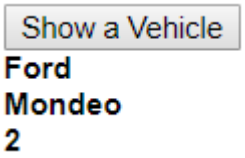
Accessing object properties & setting values

- The values can be accessed in two ways
  - objName.property or objName["property"]
  - **let** car = {make: "Ford", model: "Mondeo", age:2};

  - **let** make = car.make // make = Ford
  - **let** anoMake = car["make"] // anoMake = Ford

Simple example


```
<script>
function displayVehicles() {
   let vehicle = {make:"Ford", model:"Mondeo", age:2};
   let sHTML = "<b>";
   sHTML += vehicle.make + "<br>" + vehicle.model + "<br>" +
vehicle.age;
   sHTML += "</b>";
   clientSideContent.innerHTML = sHTML;
}
</script>
<button onClick="displayVehicles();">Show a Vehicle</button><br>
<div id="clientSideContent">Something here</div>
```



Exercise: Put into table

- Modify the previous example (slide 8) to display the contents of the single vehicle object in a table.

- Use Bootstrap to style it something like below (https://getbootstrap.com/docs/5.0/content/tables/)

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

JAVASCRIPT ARRAYS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Arrays

- Arrays can store multiple values within a single variable.
- It is a special variable that is suited for storing lists of items.
- **let** array = [“Enda”, “Barrett”];

- **let** array = [];
- You define it using square brackets

Arrays cont.

- Each item within the array is known as an element
- **let** array = [“Enda”, “Barrett”];

- Array elements can be accessed by referencing the correct index position

- **let** fullName = array[0] + “” + array[1]

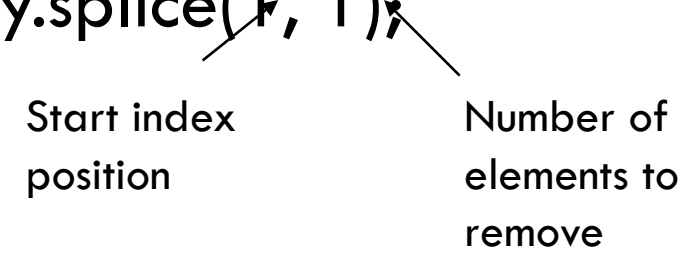
Adding array items

- When initializing an array you can add values but you may wish to add more throughout the program's execution
- This can be achieved by specifying an index position and assigning a value to that position.
- **let** array = ["Enda", "Barrett"];
- array[2] = "Peter";
- array[3] = "Devon";

Removing array items with splice

- **let** array = [“Enda”, “Barrett”];
- array[2] = “Peter”;
- array[3] = “Devon”;

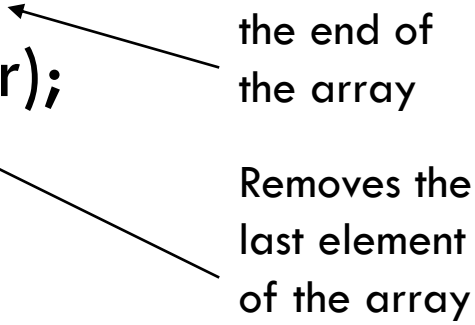
- array.splice(1, 1);



Arrays (push, pop) with Objects

- An array can also store objects
- **let** array = [];
- **let** car = {make: “Ford”, model: “Focus”, year: 2, mileage:10000}

- array.push(car);
- array.pop()



Exercise: Arrays

- Create an array of 5 objects such as vehicles etc.
- Each object has a make, model, year and mileage property.
- Populate it with mock data
- Using a loop iterate over the array of objects and place them in tabular form on your web page

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

INTRODUCTION TO NODEJS

AND SERVER-SIDE CODING

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

Overview

- Introduction to NodeJS
  - Background on NodeJS
- Where do we run it?
- Server side coding
- What does the code look like?

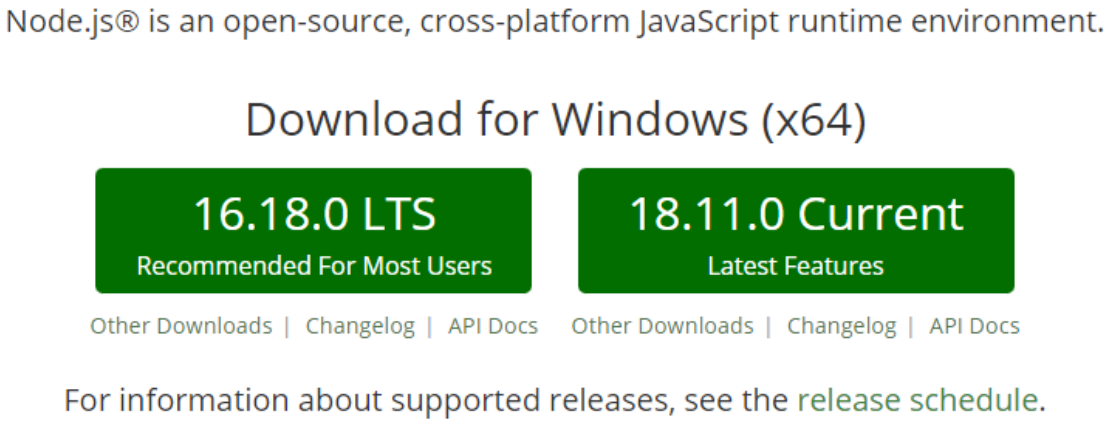
Background

- **V8** is an open-source **JavaScript engine** developed by Google. It's written in C++ and is used by the Google Chrome Browser
- Short video on V8
  - http://www.youtube.com/watch?v=hWhMKalEicY
- **Node.js** (Node) runs on the V8 engine
  - It's not a programming language – it's a runtime environment
- Write it in JavaScript
  - Most web dev's are used to writing in JavaScript

- Good News - It's just more JavaScript!

Background cont.

- It was created by **Ryan Dahl.**
  - Presented at JSConf 2009 – standing ovation
- You have downloaded it already!
- It is **Open Source.** It runs on both Linux, Mac and Windows operating systems.

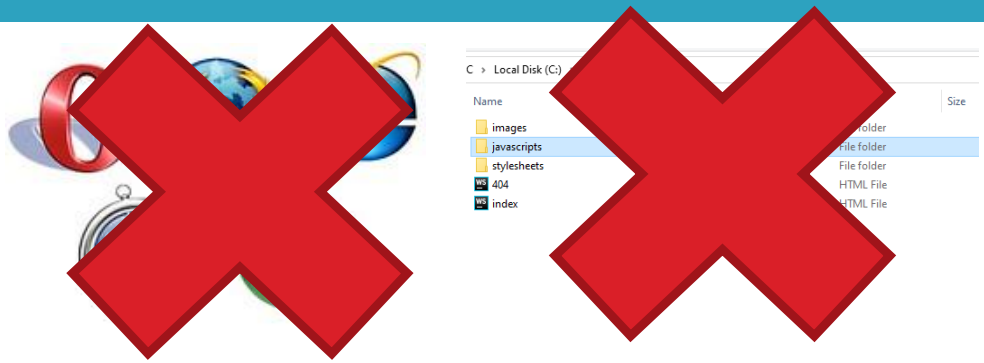


Introduction

- In simple terms Node.js is '**server-side JavaScript**'
- In not-so-simple words, Node.js is a high **performance network applications** application, well optimised for high concurrent environments
- In 'Node.js', '.js' doesn't mean that it's solely written in JavaScript. It is 40% JS and 60% C++.
- You can write modules for node in C++

Where do we run it?

- <script></script>
  - It's not for the browser



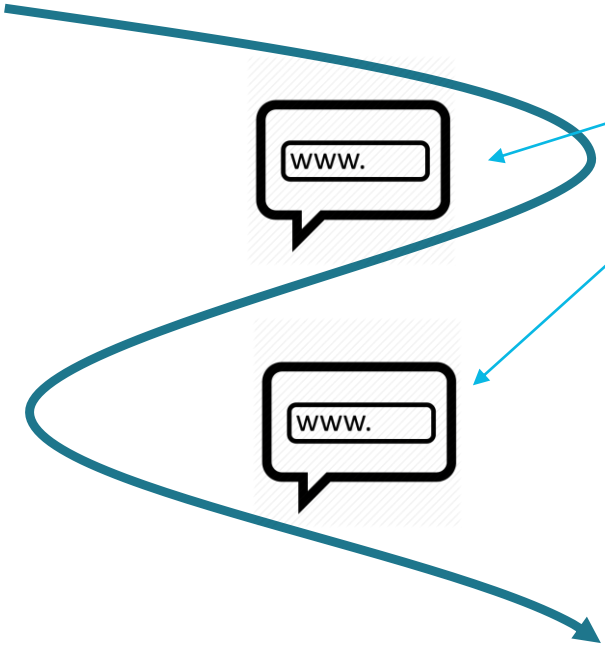
- Doesn't run on the user's laptop/phone or desktop computer!

- Where does it run?
  - On the server!

What's the server?

Bounces through the internet

Request messages from the client



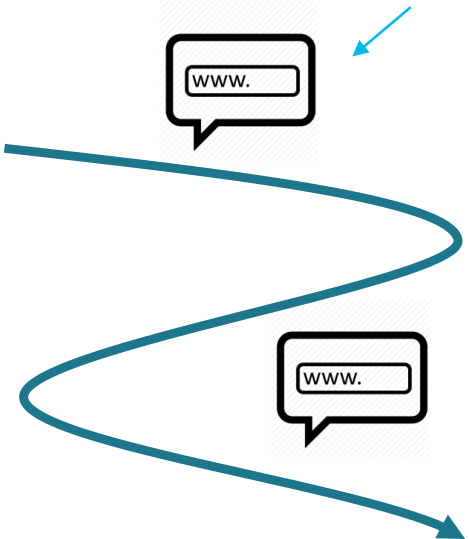
Firebase

Don't we already have this?

- Yes but at the minute all Firebase is doing is returning our static HTML pages
  - Our current HTML, CSS, and JavaScript is static, once we deploy it, it doesn't change
- Things we can't do right now
  - Save some user data to a database?
  - Register a user and store their username and password
  - Check if a user is logged in and if so allow them access to a restricted section of the site

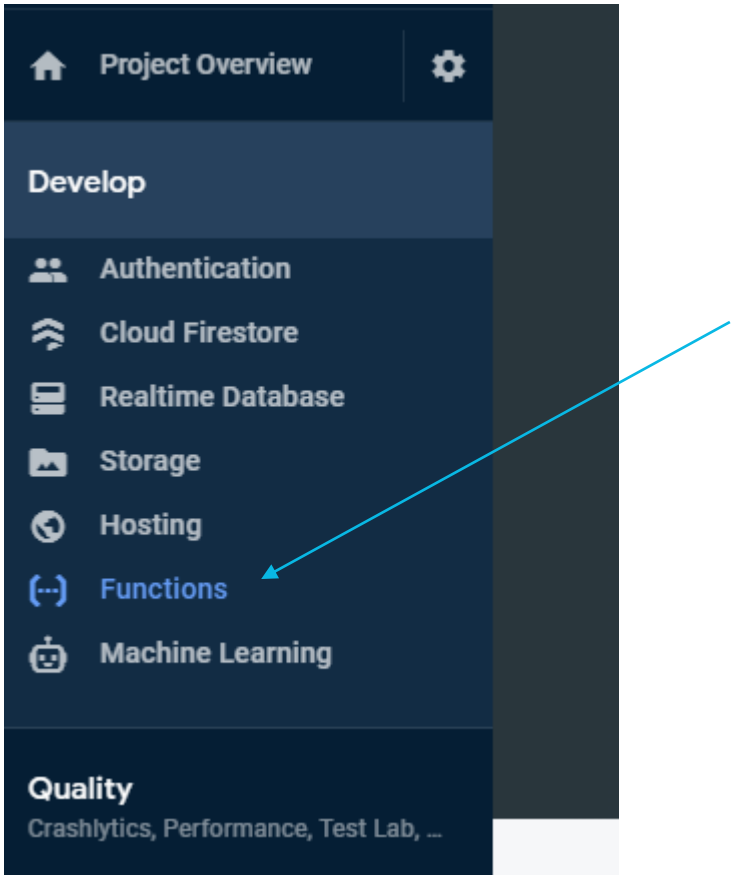
Server side coding

- In order to add the type of functionality listed on the previous slide we need to write "server side code"
  - Sometimes called backend code or a business logic layer
- It's just code that runs when the server receives a request from a client



What does the code look like?

Where does it run on Firebase



Summary

- Introduction to NodeJS
  - Background on NodeJS
- Where do we run it?
- Server side coding
- What does the code look like

REST APIS AND DEPLOYING TO FIREBASE FUNCTIONS

Overview

- Introduction to REST APIs
- Configuring Firebase functions on our apps
- Examining our first NodeJS code
- Modules in JavaScript
- Deploying our NodeJS functions to Firebase

REST API's

https://www.youtube.com/watch?v=7YcW25PHnAA&t=82s

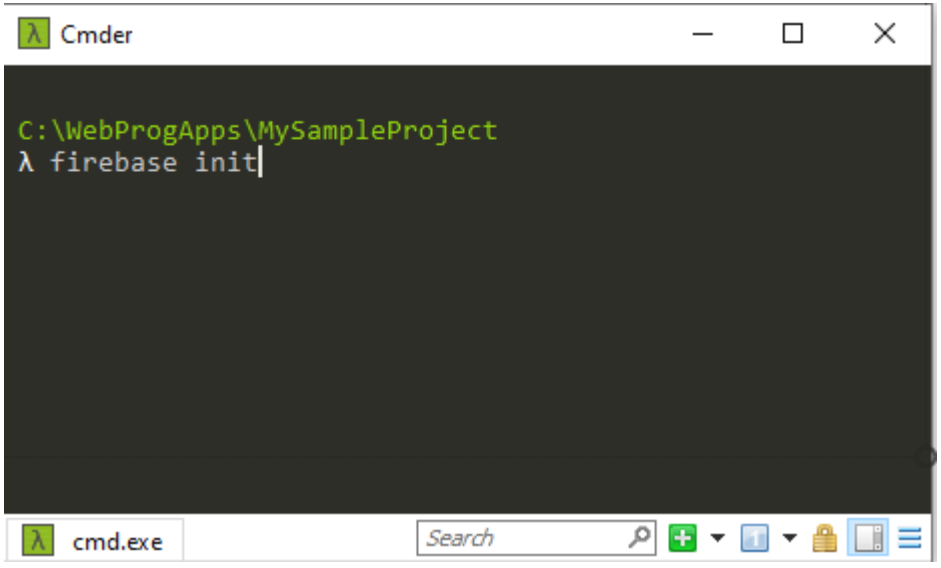
Application Programming Interface (API)

- An API expresses a software component in terms of its operations, inputs, outputs and underlying types
- RESTful APIs
  - **RE**presentational **S**tate **T**ransfer
- Use HTTP methods (verbs), GET, POST …

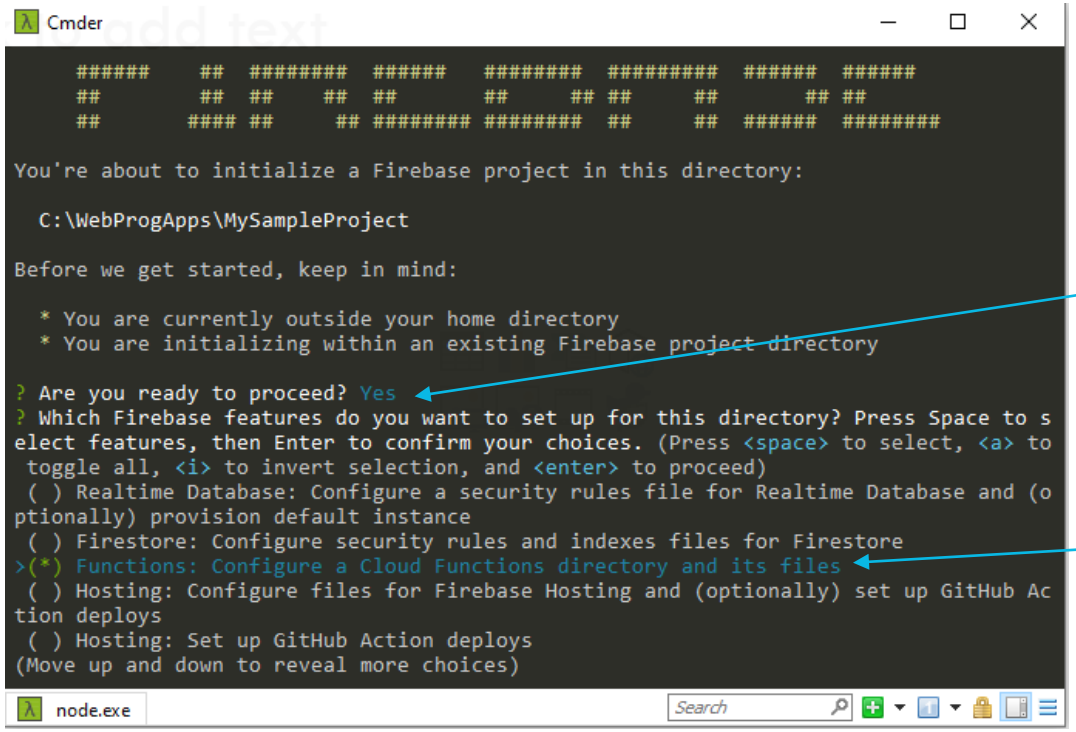
- Interface is exposed and can be invoked without caring about the underlying implementation
- Accessed via a URL

Adding Firebase Functions

- We are using Firebase functions to host our REST APIS
- Thus as when we setup hosting we must now initialise functions within our projects
- Using the command line, navigate to the directory containing your project
- firebase init



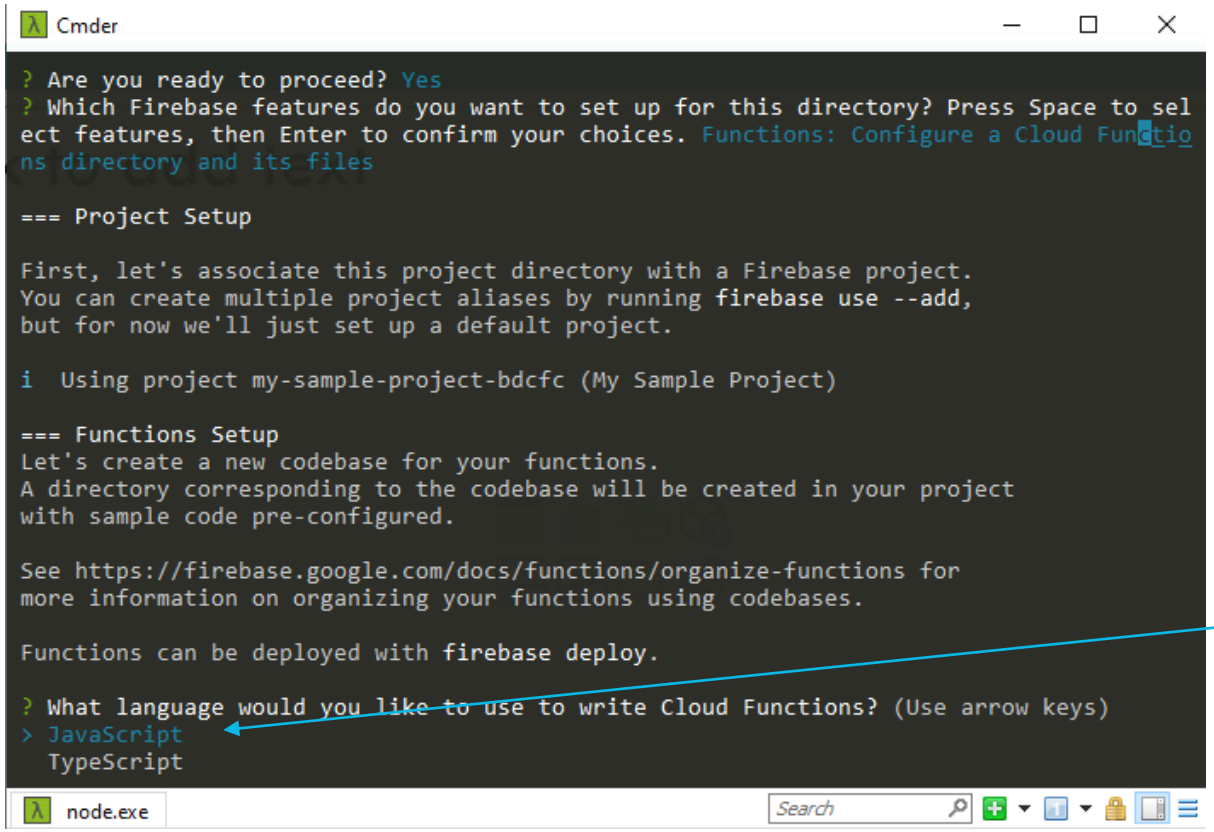
Adding Firebase Functions cont.



Select Yes

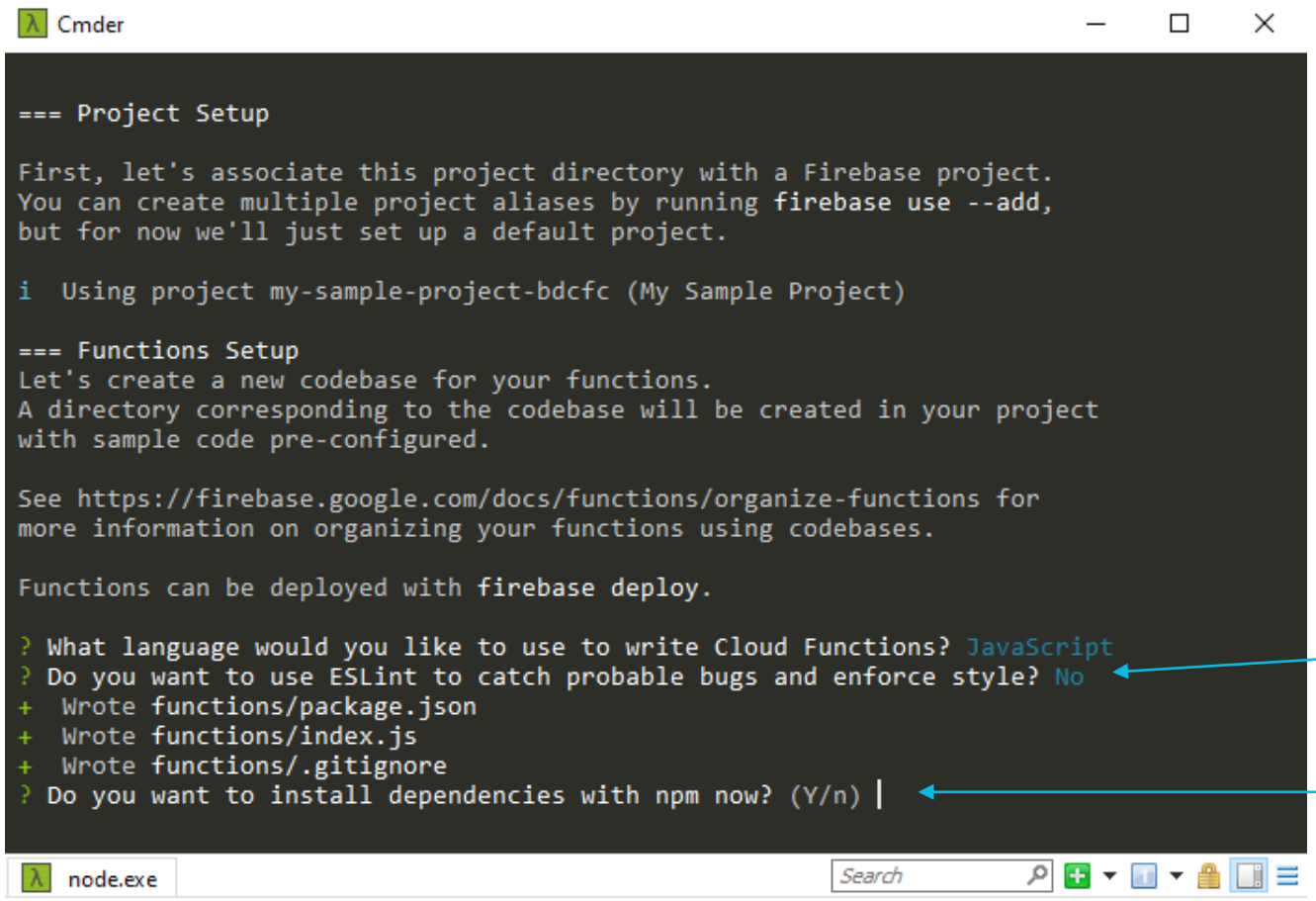
Select functions and hit enter

Adding Firebase Functions cont.



Select JavaScript

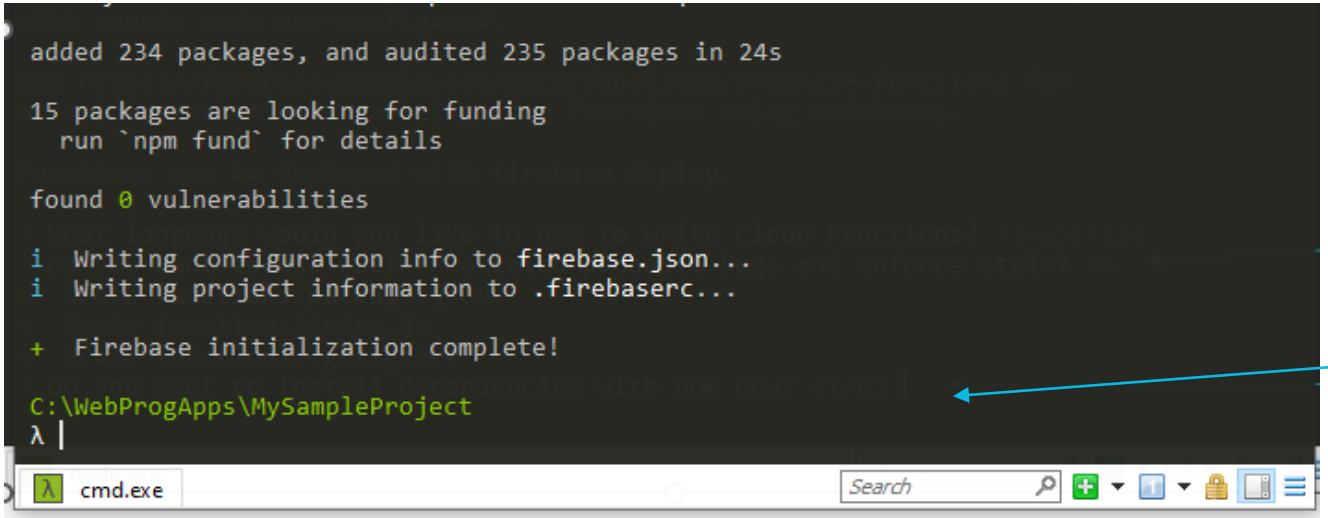
Adding Firebase Functions cont.



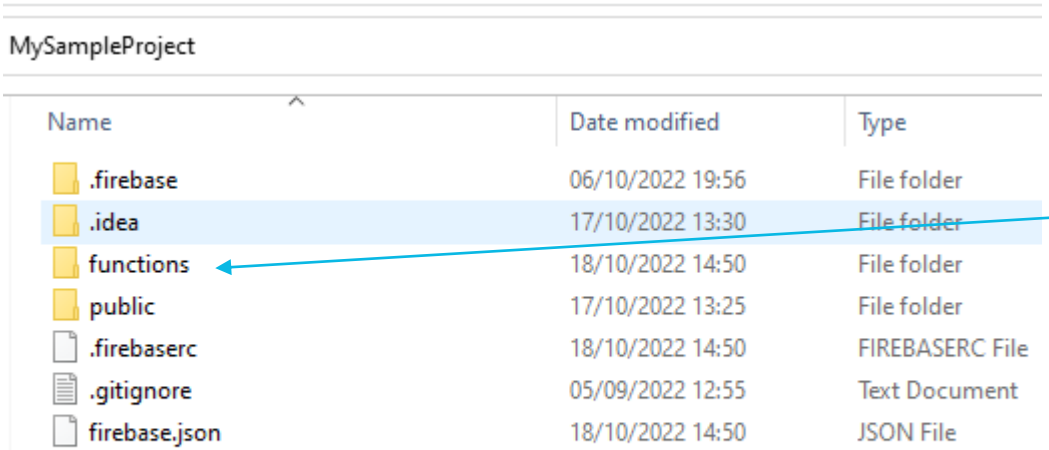
Select no to ESLint

Select Yes to installing the dependencies

Adding Firebase Functions cont.

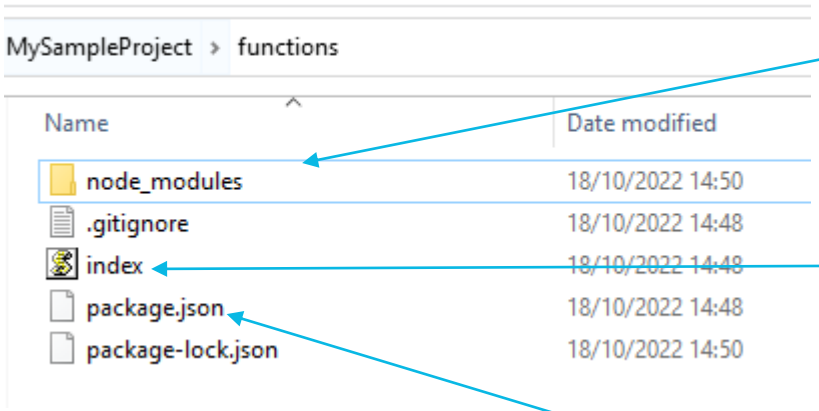


Finished installing the dependencies



Now be a new functions directory

Examining the functions folder



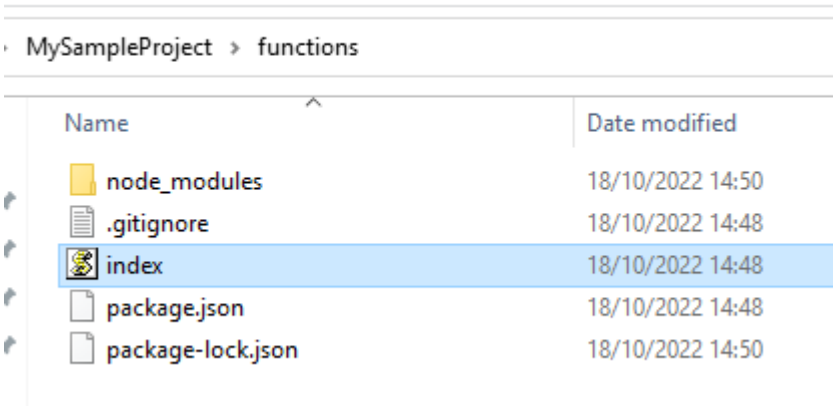
Contains dependencies

Where you will write your NodeJS code and functions

Allows you to define dependencies and other config settings

Examine the code


```
const functions = require('firebase-functions');

// // Create and Deploy Your First Cloud Functions
// // https://firebase.google.com/docs/functions/write-firebase-functions
//
// exports.helloWorld = functions.https.onRequest((request, response) => {
//   functions.logger.info("Hello logs!", {structuredData: true});
//   response.send("Hello from Firebase!");
// });
```



What is a library

- Libraries in programming are just blocks of code that you import so your program can use them?

```
require('firebase-functions');
```

- Who writes them?
  - Other Devs, yourself, could be anyone really…
- Why use them?
  - You can't use Firebase without this library.
    - It's like trying to control your TV without a remote
    - It's like trying to drive a car without a steering wheel, pedals etc.
- It's their platform so you must use their interface!

JS modules

- JavaScript allows the developer to package up code and functionality into modules

- Think of them as set of packaged functions that you wish to include in your application

JS modules

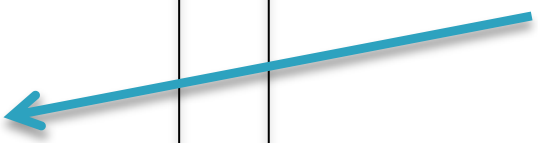
Both files in the same folder

hello.js

```
module.exports.hello = hello;

let …

function hello()
{
    return “hello”;
}
```

Driver.js

```
let mod = require('./hello');

let value = mod.hello();
```



Target

Source

Modules

- Examining our first REST API (cloud function)

```
const functions = require('firebase-functions');

// // Create and Deploy Your First Cloud Functions
// // https://firebase.google.com/docs/functions/write-firebase-functions
//
```

Exporting a module “helloWorld”

```
exports.helloWorld = functions.https.onRequest((request, response) => {
  functions.logger.info("Hello logs!", {structuredData: true});
  response.send("Hello from Firebase!");
});
```

Deploying our first cloud function

- The command is the same *firebase deploy*!
- You will see the function URL outputted to the console
- Paste it into the browser address bar to see if it works…Don't Google it! ☺

Summary

- Introduction to REST APIs
- Configuring Firebase functions on our apps
- Examining our first NodeJS code
- Modules in JavaScript
- Deploying our NodeJS functions to Firebase

FIREBASE FUNCTIONS, CALLBACKS, CREATING AND TESTING OUR FIRST FUNCTIONS

OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

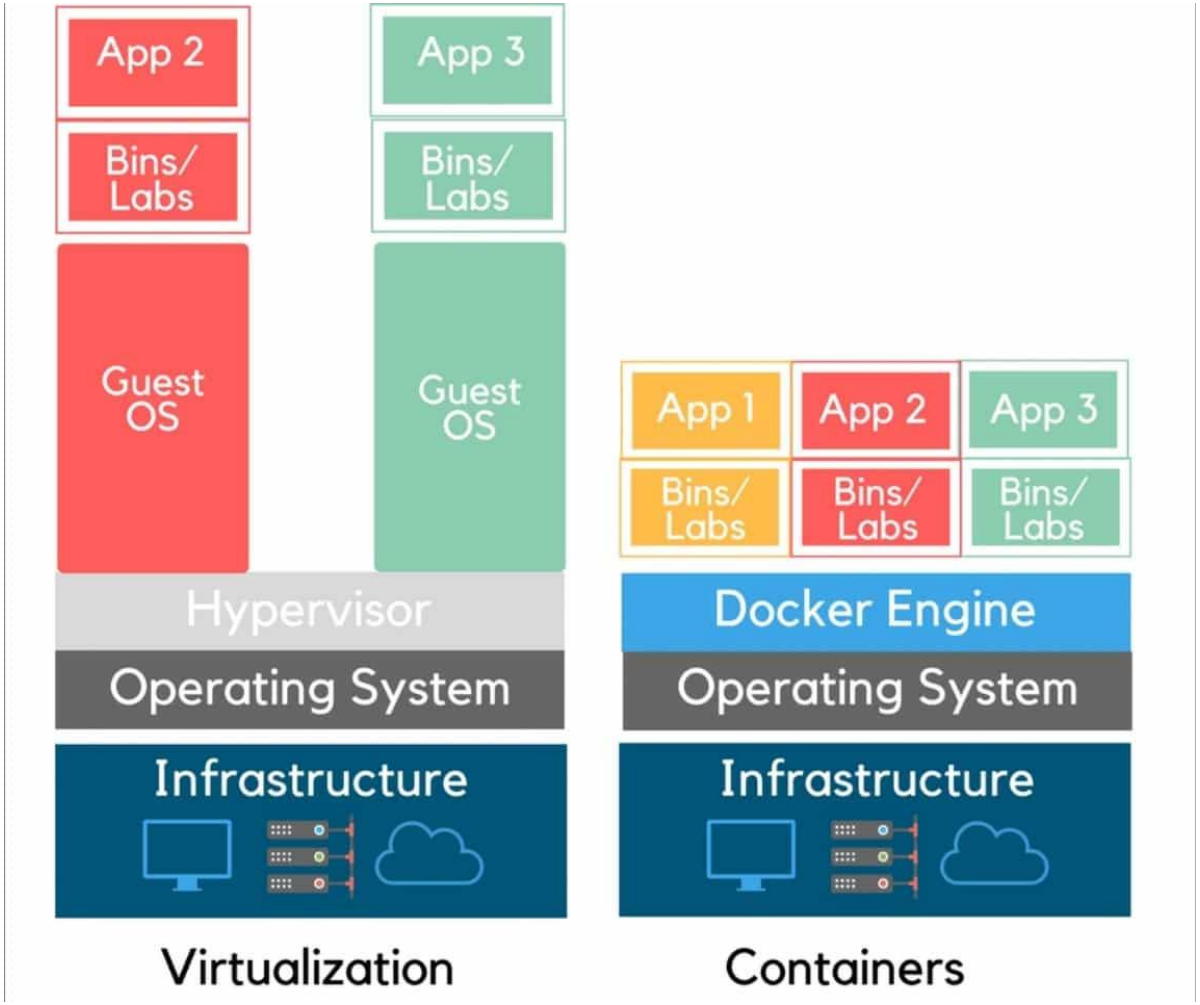
Summary

- Firebase functions
- Callback functions
- HTTP Verbs GET and POST
- Creating a dumb function which receives data and returns it back
- Testing with POSTMAN
- JSON
- Summary

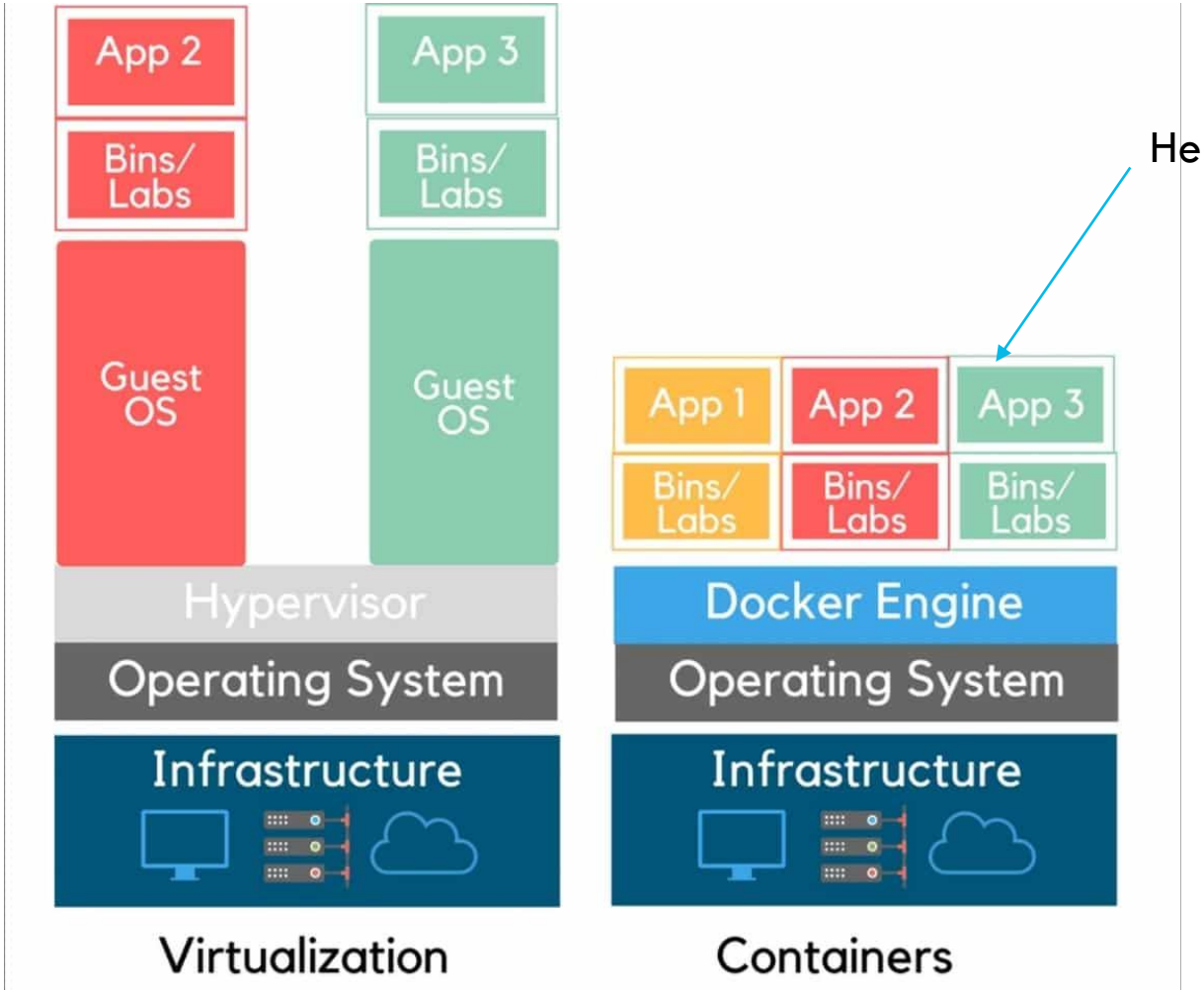
Firebase Functions

- It is a compute service that lets you run code without provisioning or managing servers.
- Similar to AWS Lambda, Azure Functions… It runs your code only when needed and scales automatically, from a few requests per day to thousands per second.
- You pay only for the compute time you consume - there is no charge when your code is not running.

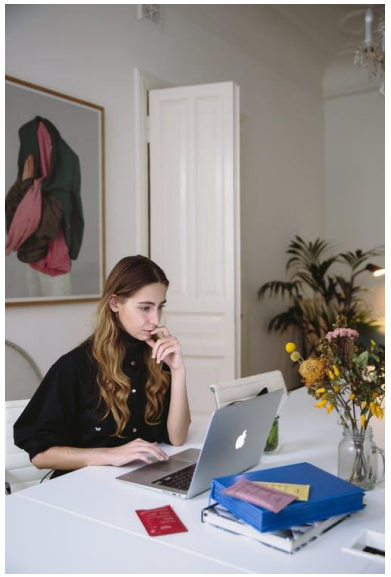
Containers vs VMs



Deploy a function what happens



Callback functions



Too busy to take a call, please leave your name and number (**function**) and I'll call you back when I'm **finished** my work (**invoke your function**).

Function callbacks

- Examining our first REST API (cloud function)

```
const functions = require('firebase-functions');

// // Create and Deploy Your First Cloud Functions
// // https://firebase.google.com/docs/functions/write-firebase-functions
// Callback function -- please run this code for me Firebase when a request is made

exports.helloWorld = functions.https.onRequest((request, response) => {
   functions.logger.info("Hello logs!", {structuredData: true});
   response.send("Hello from Firebase!");
});
```

myCoolApp/functions/index.js

Exercise: Deploying a function

- Deploy a function onto Firebase and return a string when it is invoked which says “Welcome to my cool new backend function”

- Add a second function below the first one and this time return a message saying “Not logged In!”

Passing Data - URL Query String

- We can encode parameters in the URL, we generally refer to this as the query string

- E.g. If you run a google search query, look at the URL after you search…

- http://www.mywebsite.com?data=hello

Simple function to mirror data

- Assume you want to create a function which parses out data submitted per request and mirrors it back to the requester

```
const functions = require('firebase-functions');

// Accept comment and return the same comment to the user

exports.echofunction = functions.https.onRequest((request, response) => {
    response.send(request.query.data);
});
```

myCoolApp/functions/index.js

Exercise: Parsing params from QS

- Write a function that assumes the data passed in the via the Query String is a number, take the value that is submitted per request, double it and then return it via the response.

- If the user submits a value that isn't a number then return a response that says "Please Enter a Number"

HTTP Verbs (GET and POST)

- GET and POST are HTTP request methods to transfer data from client to server.
- So far we have been making GET requests, GET is designed to request data from a specified resource
- POST is designed to submit data to the specified resource
- Both can be used to send requests and receive responses

Request Structure

- All HTTP requests have a three main parts
  - Request line
    - HTTP Method (GET, POST, etc.)
    - URL – address of the resource that is being requested
    - HTTP version
  - Headers
    - Additional information passed between the browser and the server, i.e. cookies, browser version, OS version, auth tokens, content-type
  - Message body
    - Client and server use the message body to transmit data back and forth between each other. POST request method will usually have data in the body. GET requests leave the message data empty

GET Method

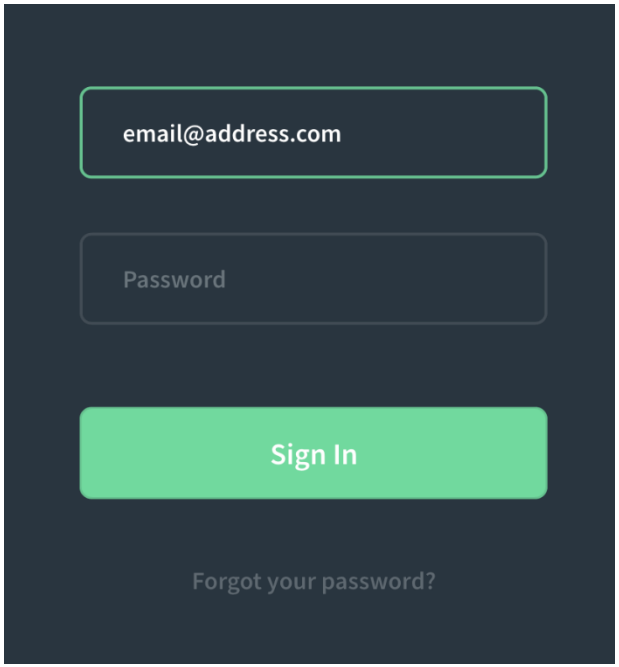
- GET requests can be cached
- GET requests remain in the browser history (you can go back!)
- GET can't be used to send binary data, like images or word documents to the server
- GET requests can be bookmarked
- GET requests have length restrictions
- GET requests should only be used to retrieve data
- Using GET data can be sent to the server by adding name=value pairs at end of the URL, i.e. Querystring
  - `mysite.app.web…/page?id=101&name=John`

POST Method

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length
- The POST method can be used to send ASCII as well as binary data

POST Request Form Data

- Form data is often sent to the server via a POST request



Alternative option is to send username and password to the server via the QueryString – uid=email@address.com
pwd=password

But is this a good idea?

GET vs POST

- Use GET if you are requesting a resource
  - You may need to send some data to get the correct response back, but in general the idea is to GET a resource

- Use POST if you want to send data to the server
- Other methods
  - PUT //Update/Replace
  - DELETE //Delete
  - PATCH //Partial update/modify

POST data to server

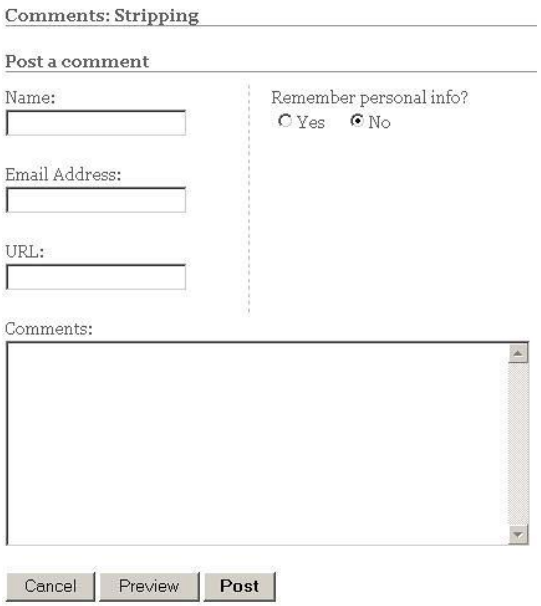
- Assume you are building a form which posts comments from your blog page to the server
- First step is to write a function to accept the comment

```
const functions = require('firebase-functions');

// Accept comment and return the same comment to the user

exports.postcomment = functions.https.onRequest((request, response) => {
    response.send(request.body);
});
```

myCoolApp/functions/index.js

How to test a POST request?

- We can create a form on a web page, then write JavaScript to send the data via POST to the server.
- However it would be nice if there was a way to test it first without having to go back to the frontend



Postman client

- When writing backend APIs such as the one we have just completed, it's often necessary to test it quickly.
- You don't want to have to write a client side request to test each API. Sometimes you may even want to pass in values which would take even longer to code up.
- Postman can help!
- https://www.postman.com/downloads/

POSTMAN

- It's brilliant for letting us test our APIs without having to write client side code to make the requests.
- It will work for all request methods, i.e. GET, POST, PUT etc.

- You can code the backend independent of the frontend!
- How else could we test to see if postcomments is working!

Sending data, what format?

- Now that we have an API available to receive data, and we have a client (postman) willing to send the data, we need to decide on a data format…

- Enter JavaScript Object Notation or JSON

JSON

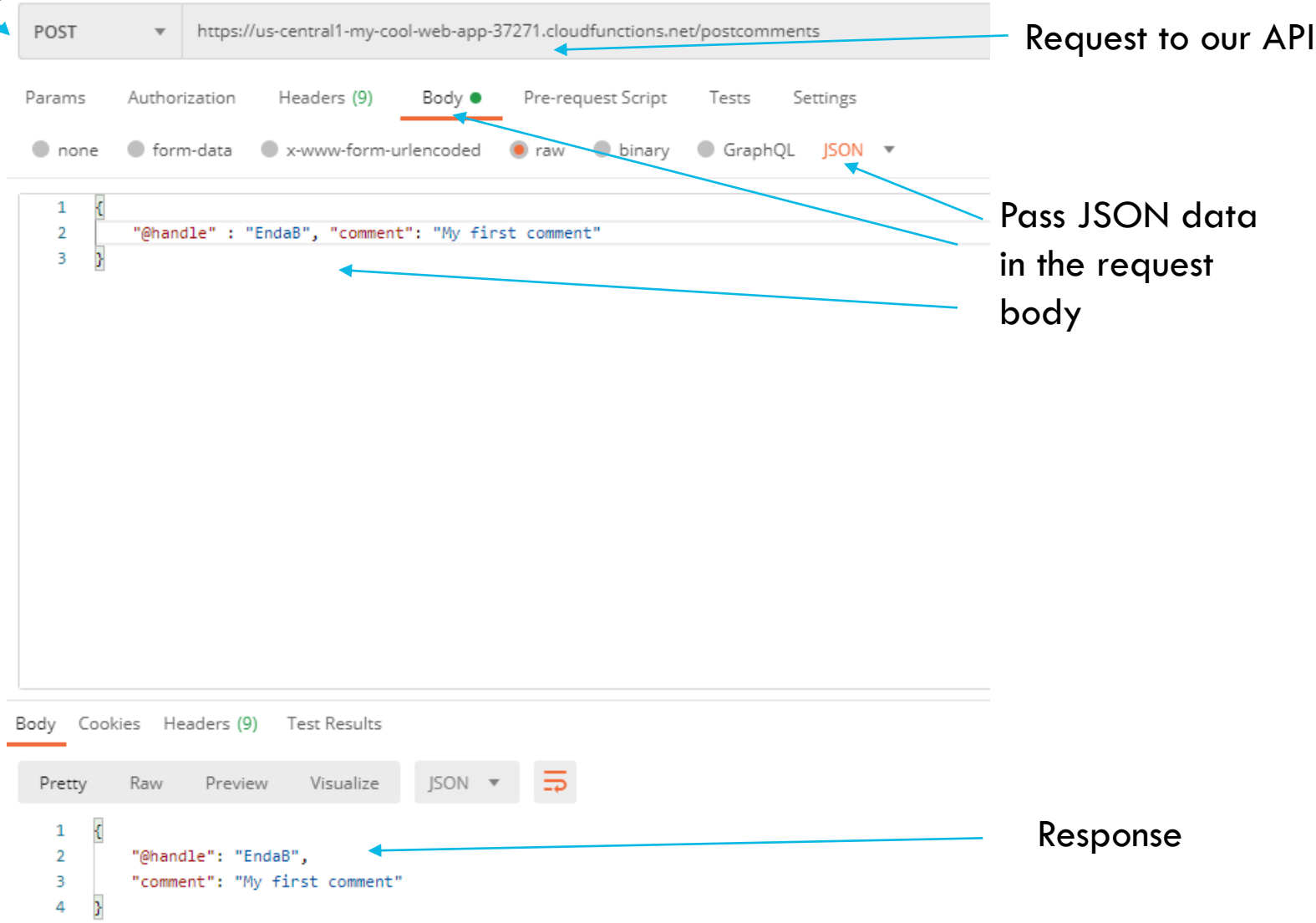
- JavaScript Object Notation (JSON)
- It is an open, human and machine-readable standard that facilitates data interchange
- Along with XML it is the main data interchange format on the web
- Data types
  - Numbers, Strings, Booleans, Arrays, Objects
  - ISODate() returns a date object
- Firestore uses JSON documents to store records of information

JSON cont.

- Arrays
  - [“a”, “b”, “c”, “d”, “e”, “f”]
  - [“apple”, 3, null, true]
- Objects
  - {“Enda” : 45, “John” : 33, “Sam” : “Smith”}
- Array of Objects
  - [{}]
- Use double quotes, no comma last value

POST JSON to Server

Set POST



Exercise 3

- Create a function which accepts comment information (JSON formatted) in the body of the request i.e. {“Comment”: “This is my comment”}
- Make a request via Postman to send the data via a POST request
- Respond with a message saying “I received your comment, thank you”.

Summary

- Firebase functions
- Callback functions
- HTTP Verbs GET and POST
- Creating a dumb function which receives data and returns it back
- Testing with POSTMAN
- JSON
- Summary

PART 1: INTRODUCTION TO FIRESTORE AND CREATING OUR FIRST DATABASE

Lecture Overview

- Firestore Database
  - Overview of Document Driven Databases
  - Creating our first database
- Connecting the database to our Firebase functions
  - Writing our comment data to the database
  - Reading our comment data from the database
- All will be tested using POSTMAN

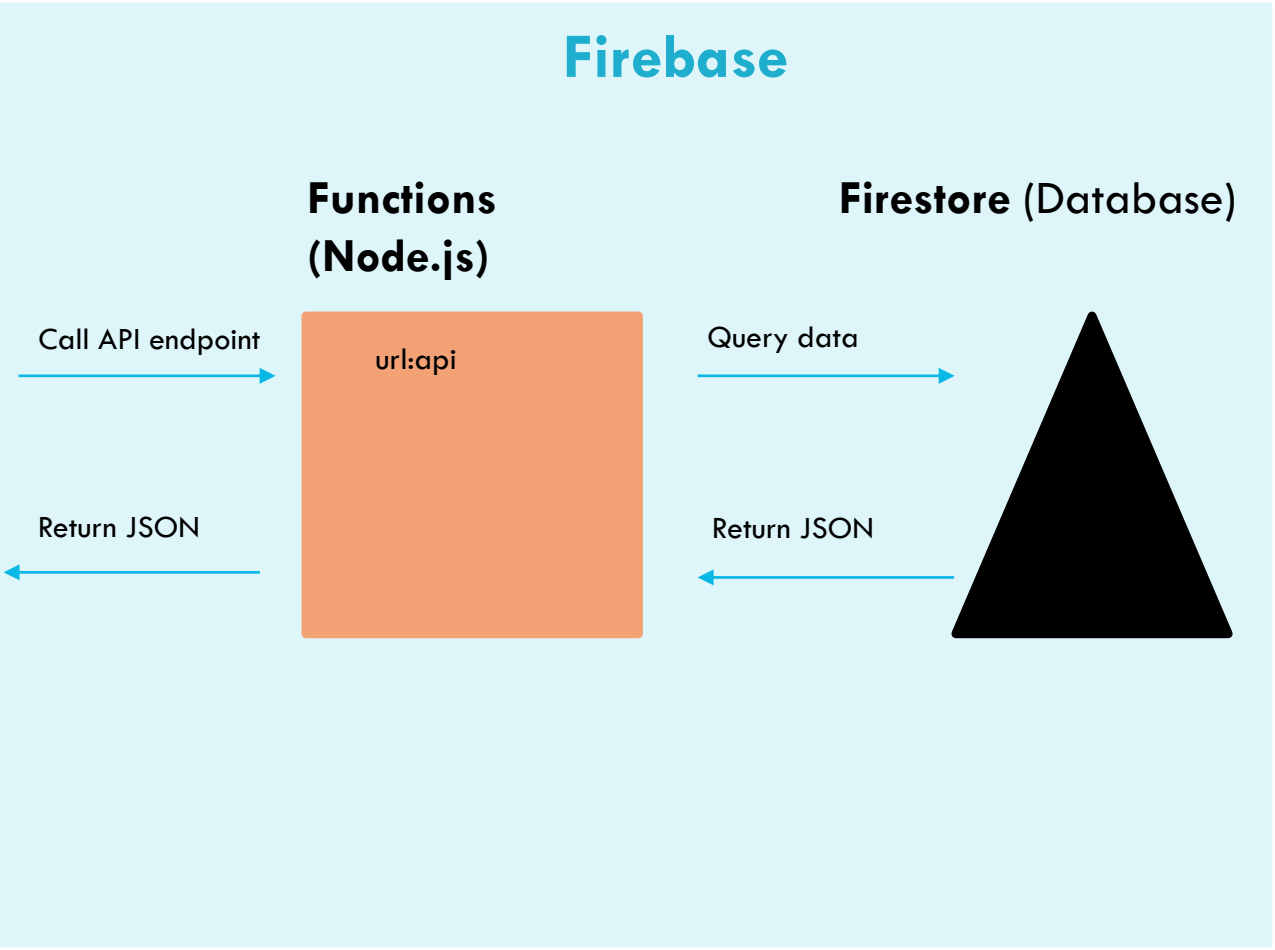
Purpose of the lecture

- The goal is to introduce you to Firestore, from the point of view of using it as a backend for your applications. The majority of the discussion will be practically focussed, with little theory concerning more advanced database concepts such as sharding, normalisation, concurrency, BSON, locking writes/reads etc.

- It will be a basic introduction on how to get a database connected to your applications.

Architecture

Clients



What is Firestore?

- Firestore is a Document Driven Database.
- Documents follow a **property:value** format
  - JSON
- Scalable, highly performant and document oriented.
- The databases tend to scale more easily horizontally.

Database concepts

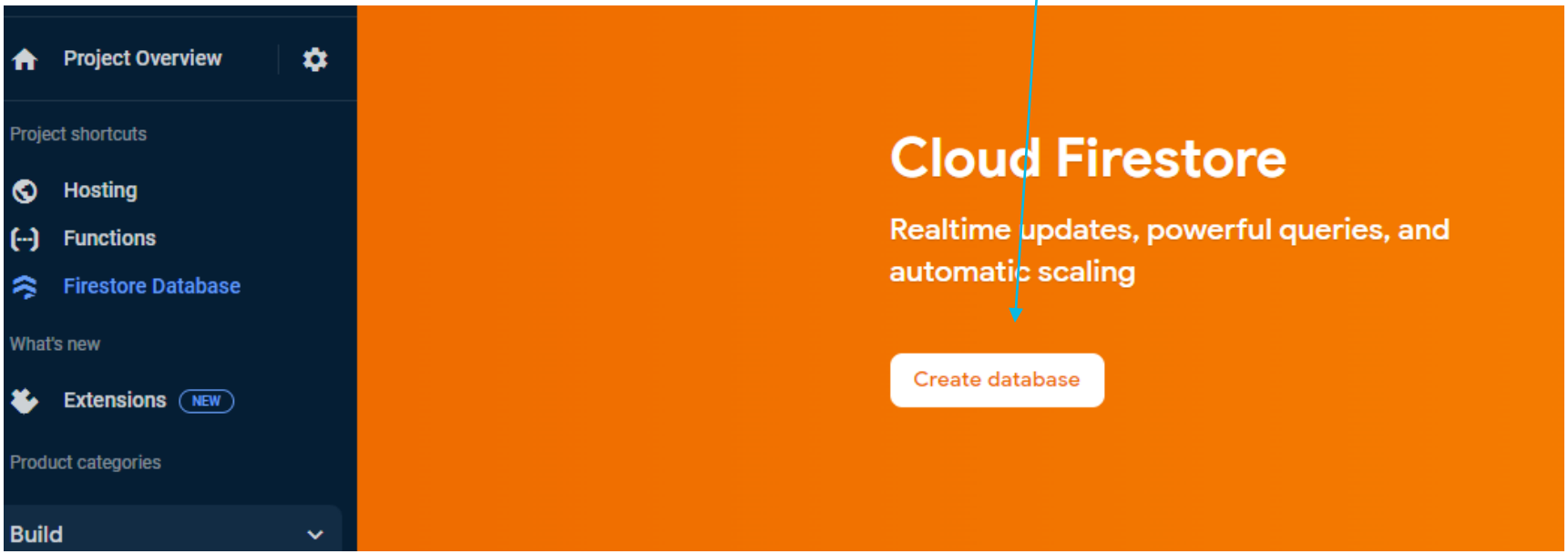
- Records in Firestore are known as "**Documents**"
  - These *documents* are just JSON data
- Documents are grouped into **"Collections"** which are equivalent to tables in relational databases
- Queries are still queries, however there is NoSQL!

SQL to Firestore Terminology

Database	⟷	Database
Table	⟷	Collection
Record/Tuple/Row	⟷	Document
Column	⟷	Field

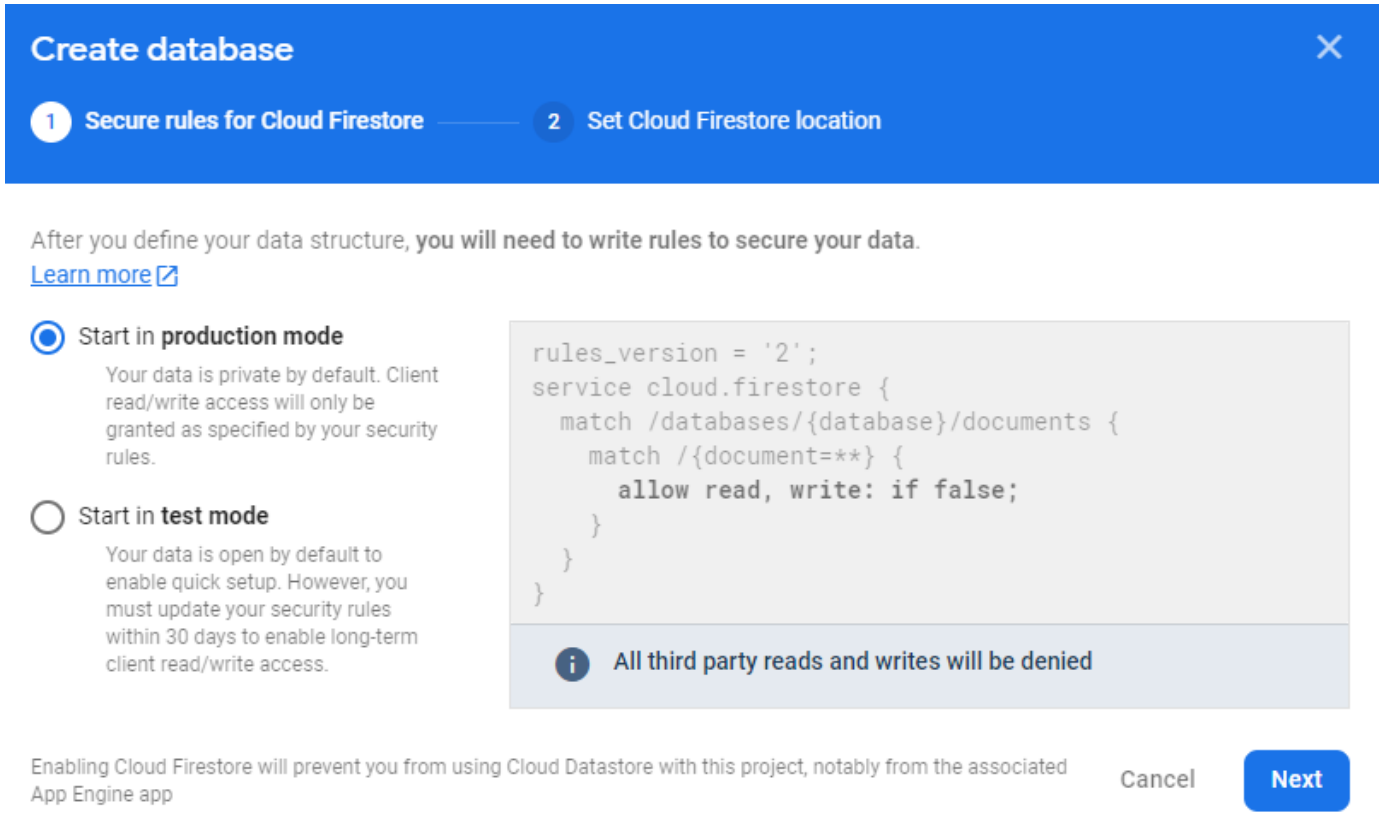
Creating our first database

Login to the Firebase dashboard, click on Firestore and then “Create database”



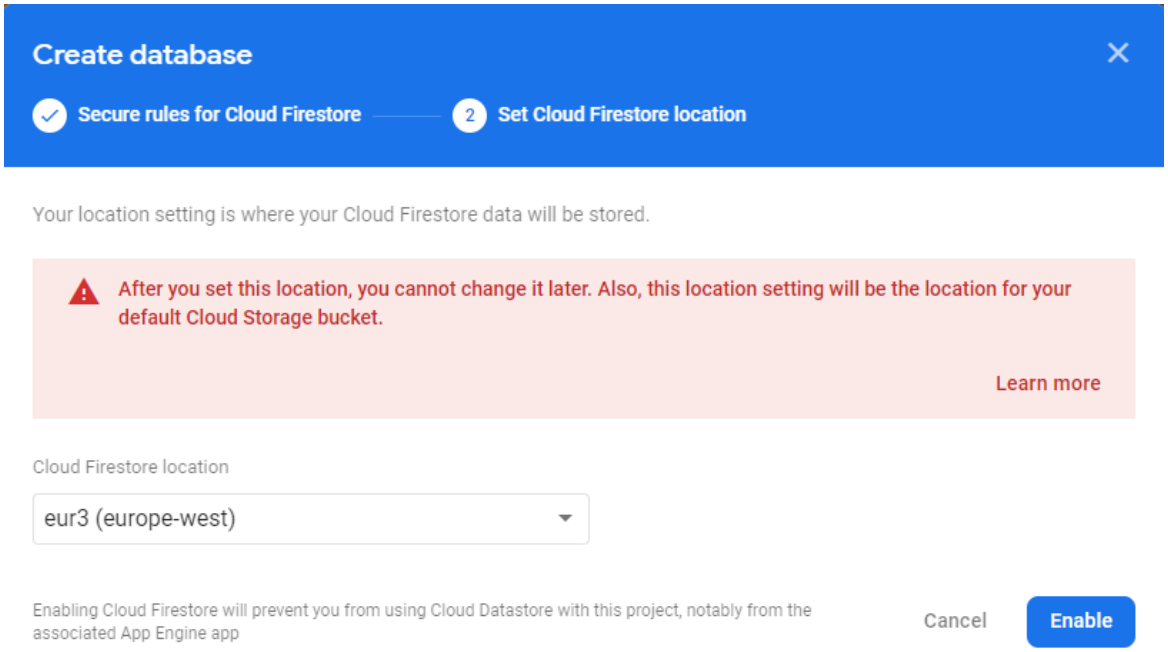
Open in production mode

- Start in production mode



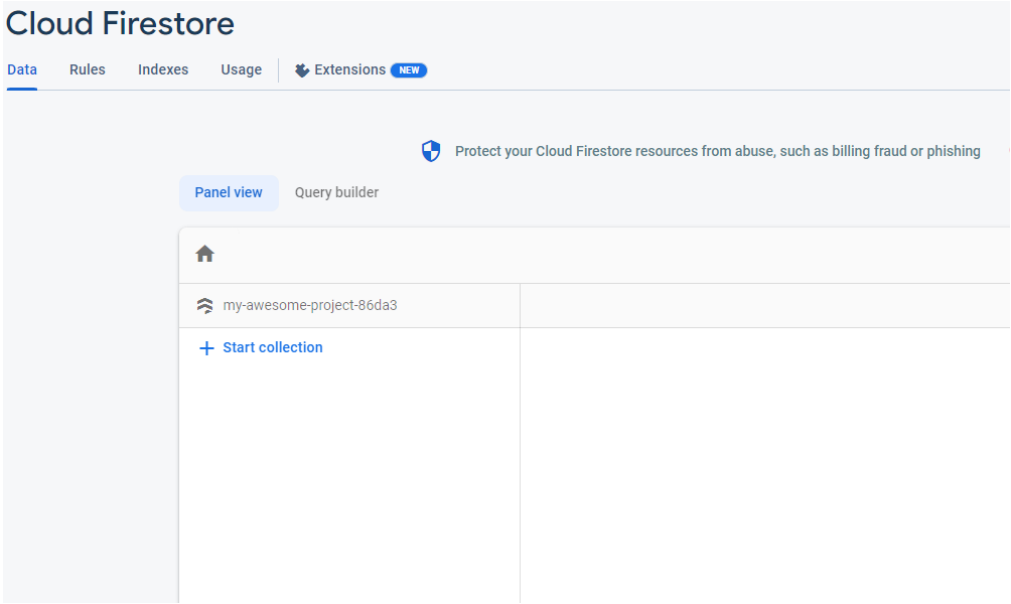
Choosing a region

- The latency should be fairly low so the default region will be fine, but if you want to place it in Europe please select it in the dropdown and then click enable



Database is now created

- You can create a collection and add documents manually via this web interface. But the next step is to connect to it with our functions and read/write data.



Summary Overview

- ~~Firestore Database~~
  - ~~Overview of Document Driven Databases~~
  - ~~Creating our first database~~

- Connecting the database to our Firebase functions
  - Writing our comment data to the database
  - Reading our comment data from the database

Writing data to the database

- To motivate data writing we will reuse the postcomments function
- This is known as “Creating” a document
- I’ll create a new document every time the postcomments function is called and save it in the database
- https://firebase.google.com/docs/firestore

Firebase admin

- Firebase provides an admin library to allow your server code (functions) to run in an authenticated mode

- This means your code can connect to the database, create docs, delete docs, update etc. all securely

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();
```

Promise – More async hell

- In ES6 a new concept was added to JavaScript to handle **Callback hell**
- These are called promises
- What's the difference between callbacks and promises?
  - Callback is passed as an argument
  - Promise is something that is achieved or completed in the future.
    - Promise is an object, **then()** method (if promise is fulfilled) and **catch** (if promise is rejected)

Code examples


```
asyncFunc(result => {
    console.log(result);
});
```

Callback

```
const promise = asyncFunc(()=>{
    return new Promise...
});
promise.then(result => {
    console.log(result);
});
```

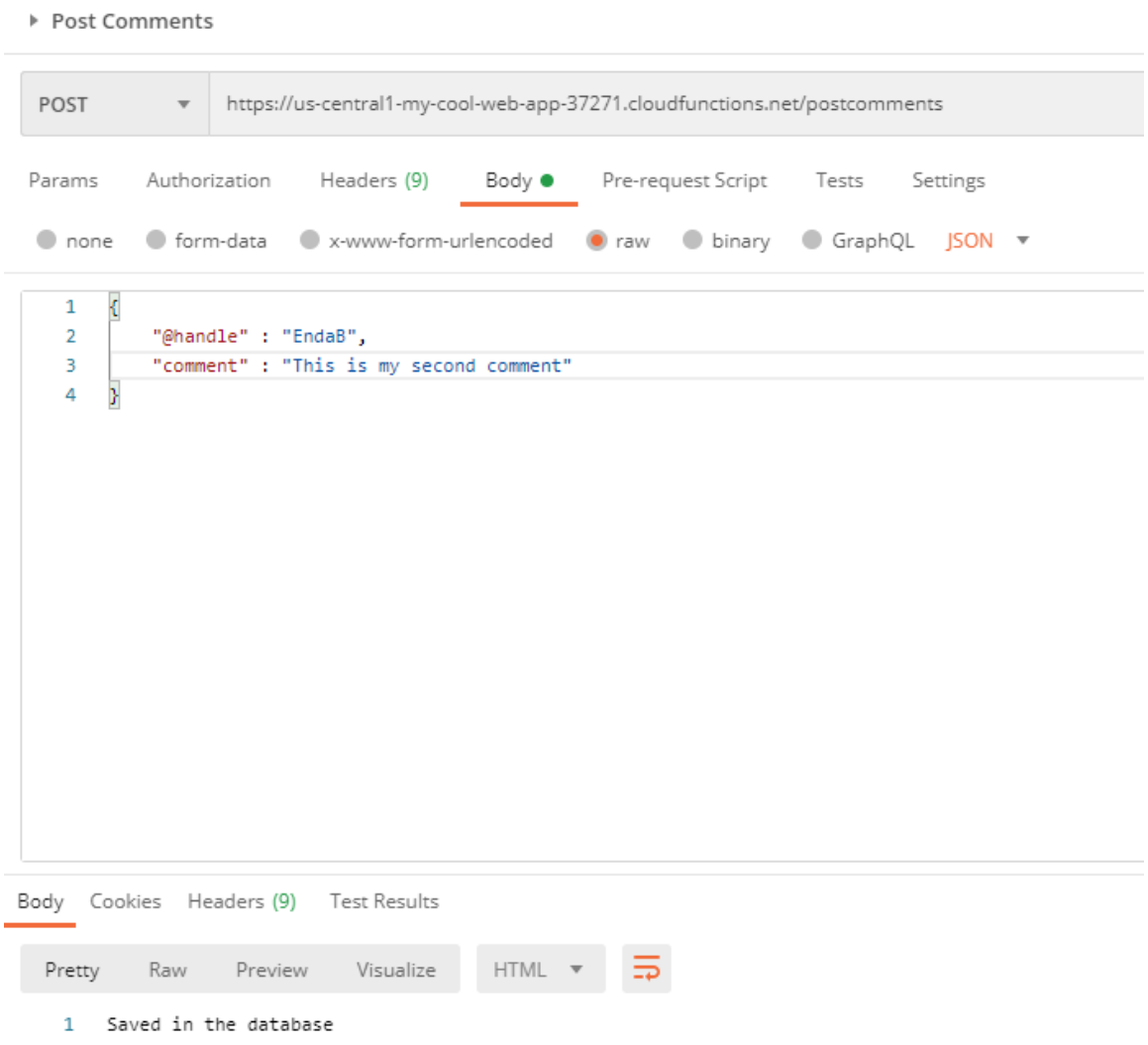
Promise

Adding a document


```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.postcomments = functions.https.onRequest((request, response) => {
   // 1. Receive comment data in here from user POST request
   // 2. Connect to our Firestore database
   return admin.firestore().collection('comments').add(request.body).then(()=>{
                        response.send("Saved in the database");
   });
});
```

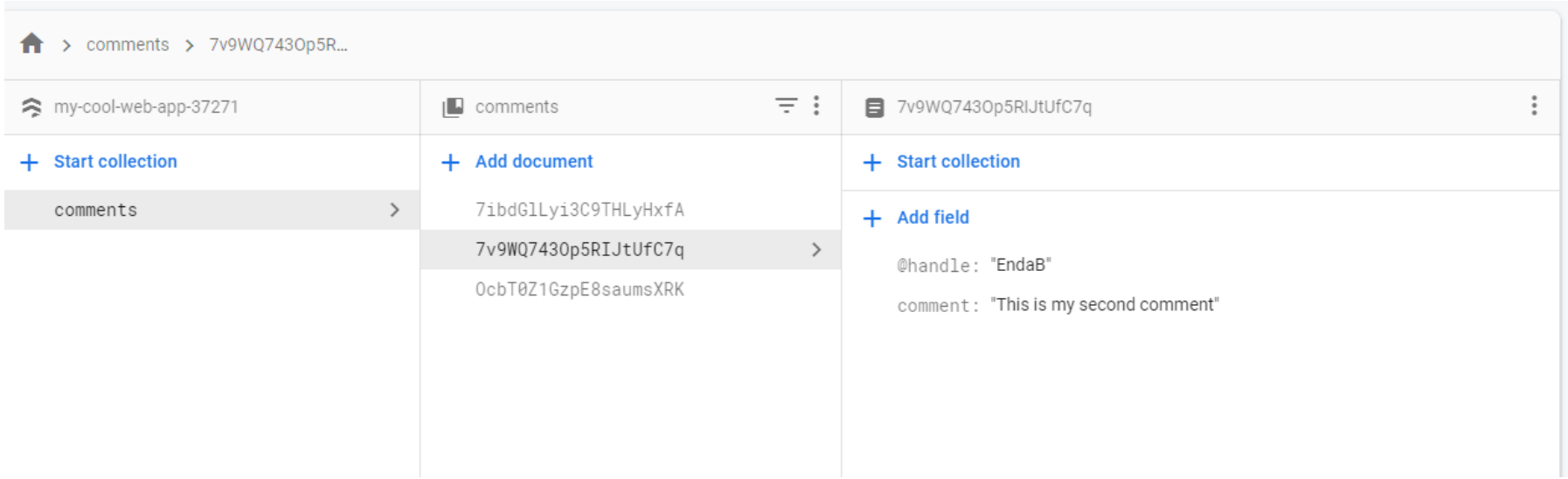
myCoolApp/Functions/index.js

Using POSTMAN POST to the fn



Check the database to see if it saved

- If you check on Firebase you should now see your comment

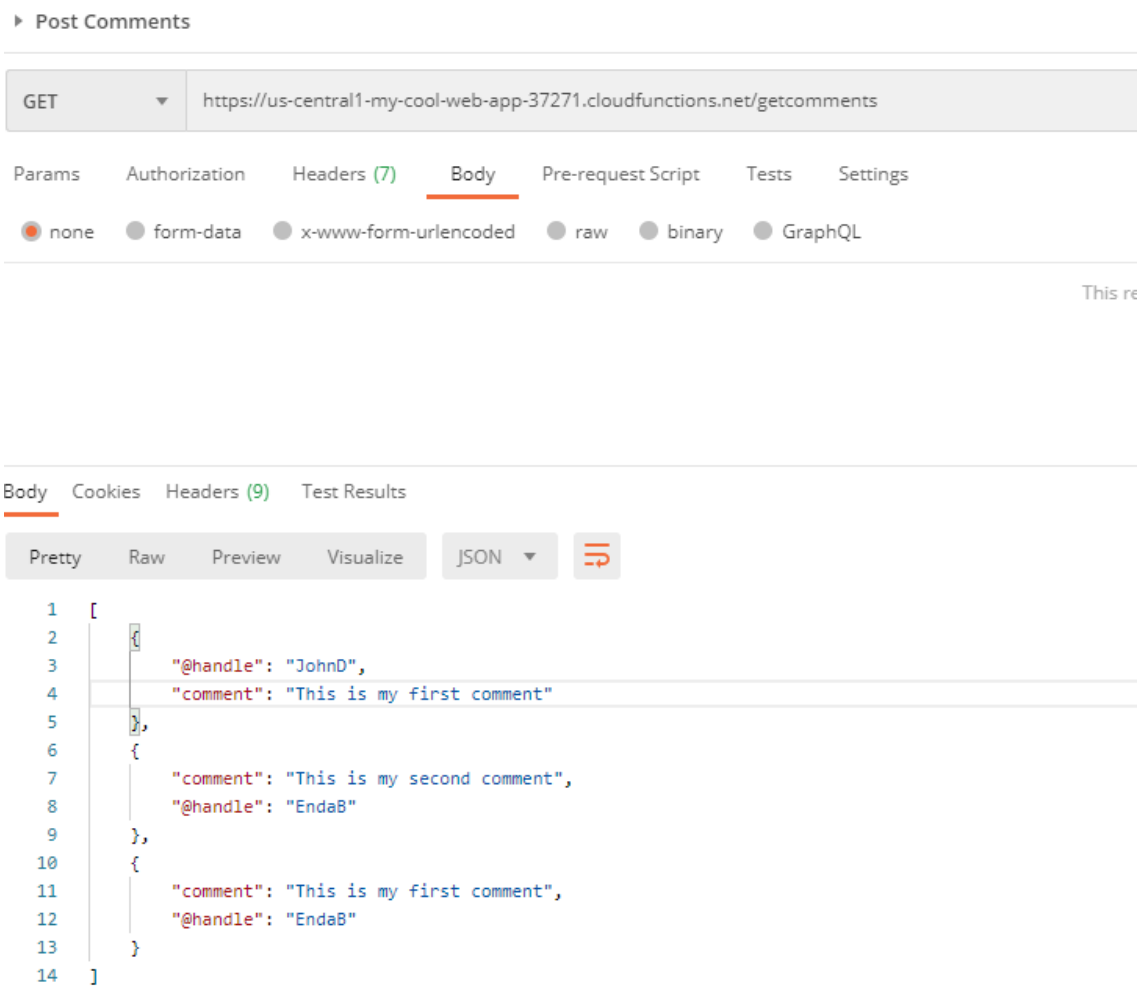


Reading our documents


```
exports.getcomments = functions.https.onRequest((request, response) =>
{

    // 1. Connect to our Firestore database
    let myData = []
    admin.firestore().collection('comments').get().then((snapshot) => {

                if (snapshot.empty) {
                        console.log('No matching documents.');
                        response.send('No data in database');
                        return;
                }

                snapshot.forEach(doc => {
                        myData.push(doc.data());
        });

        // 2. Send data back to client
        response.send(myData);
    })
});
```

myCoolApp/Functions/index.js

Test the function with POSTMAN



```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.postcomments = functions.https.onRequest((request, response) => {

   // 1. Receive comment data in here from user POST request
   // 2. Connect to our Firestore database
   admin.firestore().collection('comments').add(request.body);
   response.send("Saved in the database");
});

exports.getcomments = functions.https.onRequest((request, response) => {

   // 1. Connect to our Firestore database
   let myData = []
   admin.firestore().collection('comments').get().then((snapshot) => {

                    if (snapshot.empty) {
                              console.log('No matching documents.');
                              response.send('No data in database');
                              return;
                    }

                    snapshot.forEach(doc => {
                              myData.push(doc.data());
          });

          // 2. Send data back to client
          response.send(myData);
   })
});
```

myCoolApp/Functions/index.js

OrderBy

- So far when reading comments from the database we have not given any consideration to their order
- Perhaps it would be useful to order them by postdate or perhaps by the number of likes etc.
- To do this we need to modify our Firebase functions postcomments and getcomments to order the comments

Creating comments - postcomments

- The Firestore database supports a timestamp field, which we can use to store the date and time each comment was posted.

- Once this is recorded on each document we can return the comments to the user in order of their post date/time.

Posting comments

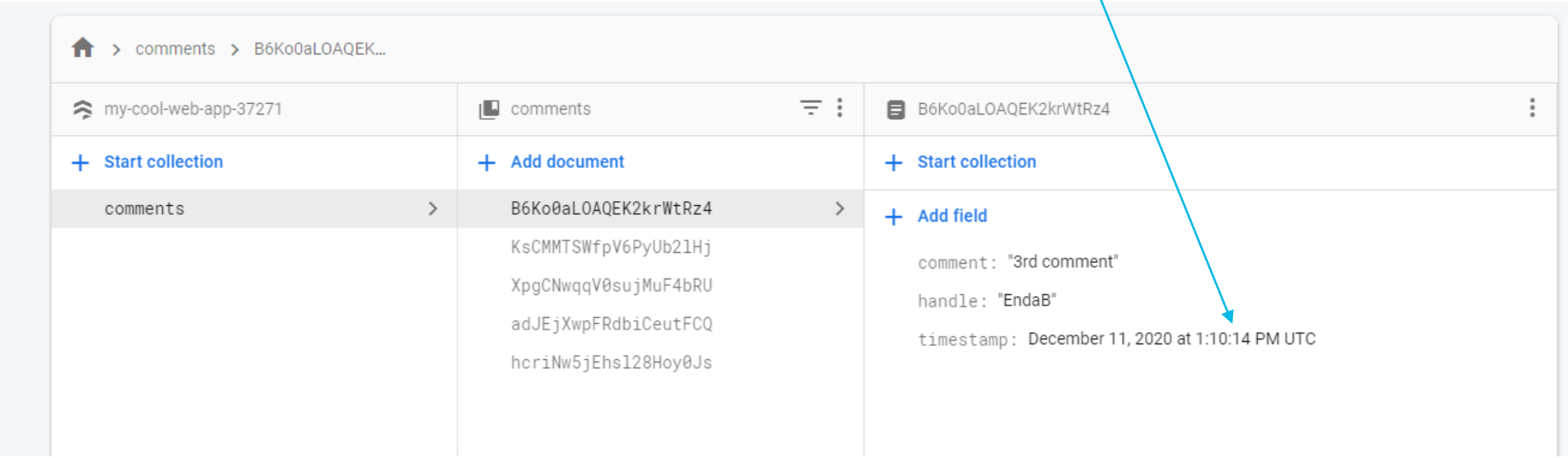

```
exports.postcomment = functions.https.onRequest((request, response) => {
        console.log("Request body", request.body);
                // Create a timestamp to add to the comment document
        const currentTime = admin.firestore.Timestamp.now();
        request.body.timestamp = currentTime;

        admin.firestore().collection('comments').add(request.body).then(()=>{
                response.send("Saved in the database");
        });
});
```

Don't forget to hit *firebase deploy* once you have made your changes

myCoolApp/functions/index.js

Check database

- When you post a comment you should now see a timestamp beside each comment



Ordering documents by timestamp

- We now modify the get comments firebase function to order the comments by timestamp

```
exports.getcomments = functions.https.onRequest((request, response) => {

    // 1. Connect to our Firestore database
    let myData = []
    admin.firestore().collection('comments').orderBy('timestamp').get().then((snapshot) => {

                        if (snapshot.empty) {
                                    console.log('No matching documents.');
                                    response.send('No data in database');
                                    return;
                        }

                        snapshot.forEach(doc => {
                                    myData.push(doc.data());
            });

            // 2. Send data back to client
            response.send(myData);
    })
});
```

myCoolApp/functions/index.js

Lecture Overview

- Firestore Database
  - Overview of Document Driven Databases
  - Creating our first database
- Connecting the database to our Firebase functions
  - Writing our comment data to the database
  - Reading our comment data from the database

SOURCE CONTROL – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

Ollscoil na Gaillimhe
University of Galway

Source control - overview

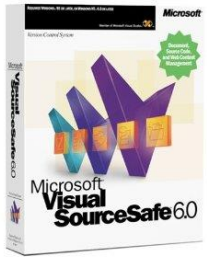
- Version (Source) control
  - Version control software
  - Why we need it?
- Git
  - What is Git
  - Branching
  - Pull requests

Version control

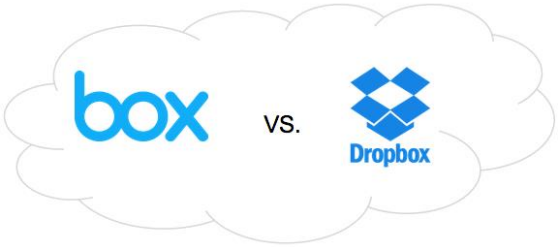
- What is version control?
  - **Version control** is a system that records changes to a file or set of files over time so that you can recall specific **versions** later
  - Also known as **revision control** or **source control**
- Keeps track of changes, by whom and when
- Fundamental tool for developing software projects

Version control software?

- Subversion, GIT, VSS, CVS, Mercurial



- Revision control is actually present in a variety of software products

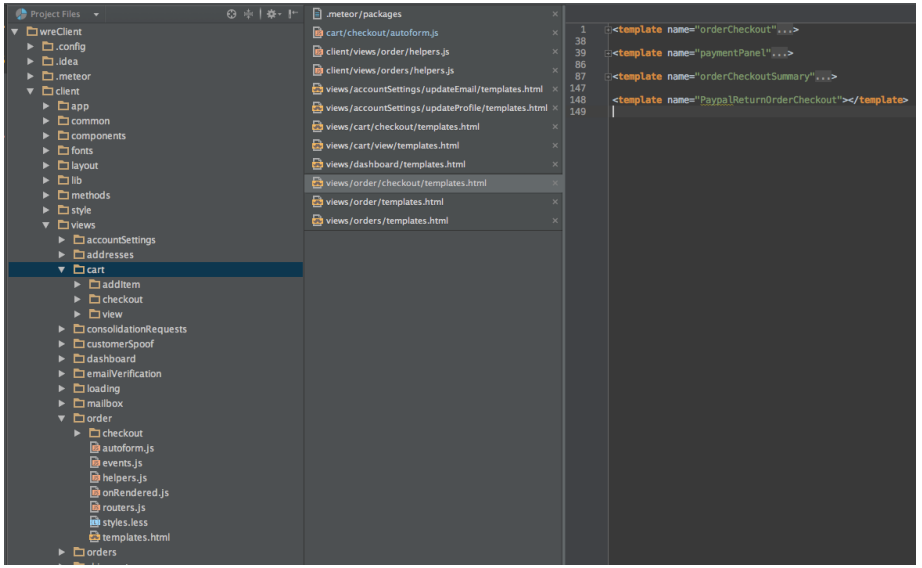


Why do we need it?

- Backup software source
  - Roll back to previous versions
- Keeping a record of who did what and when
  - Know who to praise and who to fire!
- Collaborating with other people (teams)
- Troubleshooting
  - Analyse the change history to figure out what caused the problem
- Statistics
  - Find out who is the most productive!

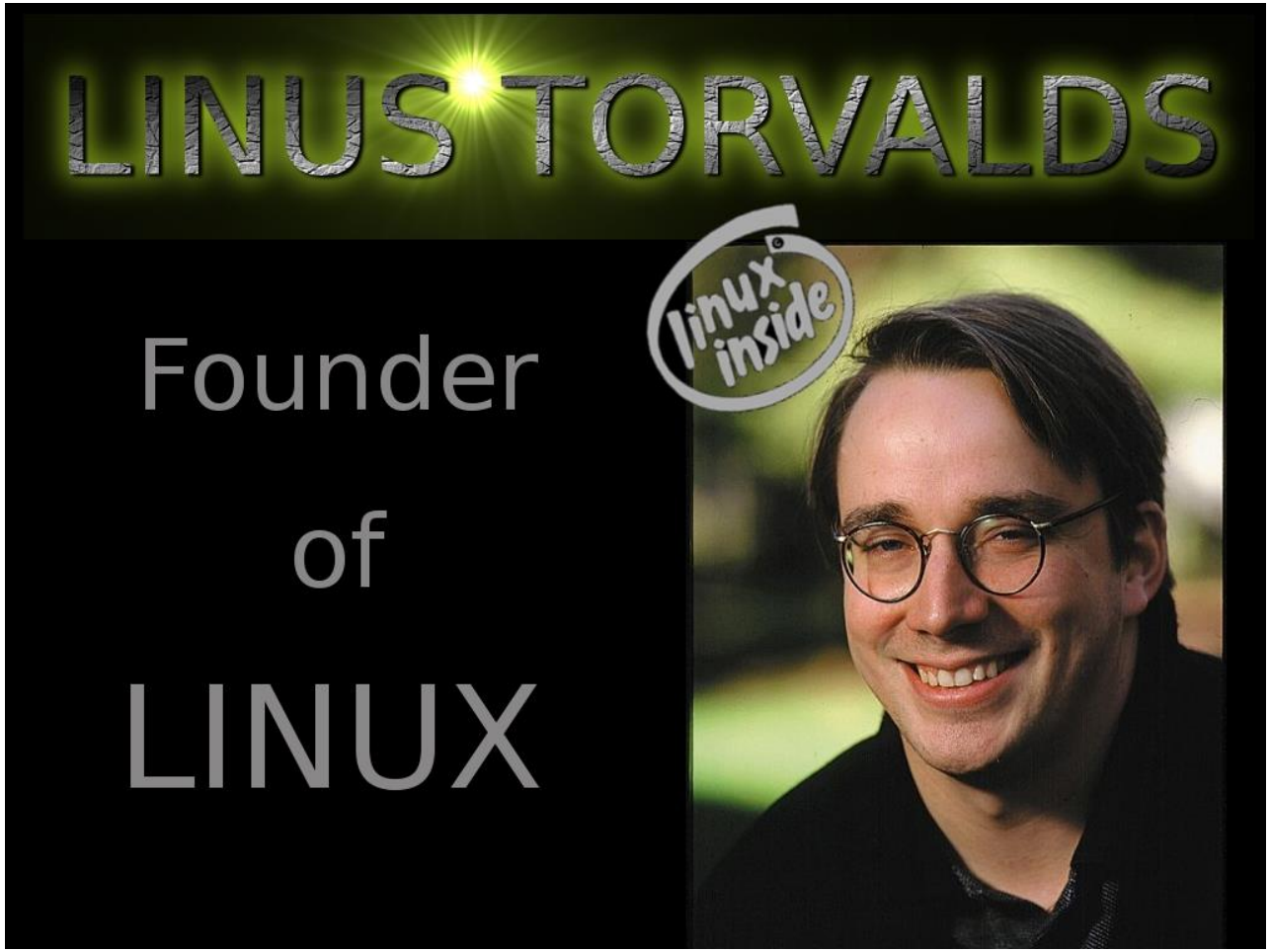
What should you commit?

- Web project (HTML, CSS, JavaScript, Images, Documentation, Functions, Configuration files)



- Not the Node Modules folder

GIT Creator



Why was it created?

- For a long time Torvalds wasn't using any version control for the Linux Kernel (1991-2002).
- Changes were passed around as patches and archived files.
- In 2002 they began using BitKeeper for managing the source for the Linux Kernel
- In 2005 the relationship broke down and BitKeeper revoked their licence.
- Initial release 7th April 2005

https://git-scm.com/book/az/v1/Ba%C5%9Flan%C4%9F%C4%B1c-A-Short-History-of-Git

They needed something similar to BitKeeper

- The developers had the following objectives in mind:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)

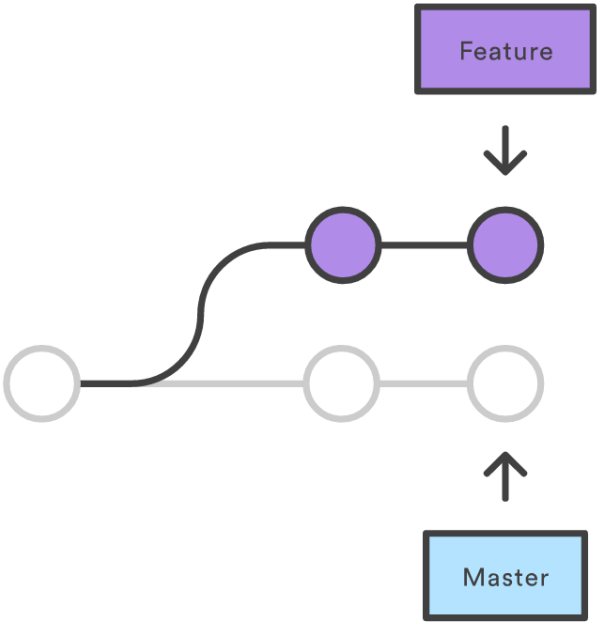
Why is everyone moving to GIT?

- Not all but quite a few!
- One of the most touted reasons is that of DVCS
  - Works great when you have no access to the internet! No VPN access to the SVN server…
  - No single point of failure
- Even offline you can access the history, branches, versions etc…

Feature branch workflow

- It encourages branching for every feature

- No matter how big or small the feature, a branch can easily be created and is encouraged.



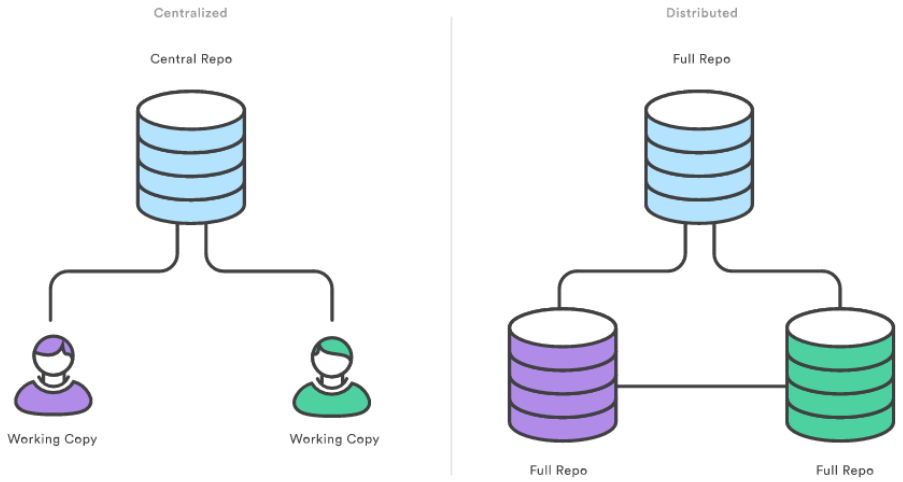
https://www.atlassian.com/git/

Distributed development

Each developer gets their repo complete with history of commits

This makes GIT extremely fast
You don't need a network connection to

- Commit changes
- Inspect previous versions
- Perform diffs between commits

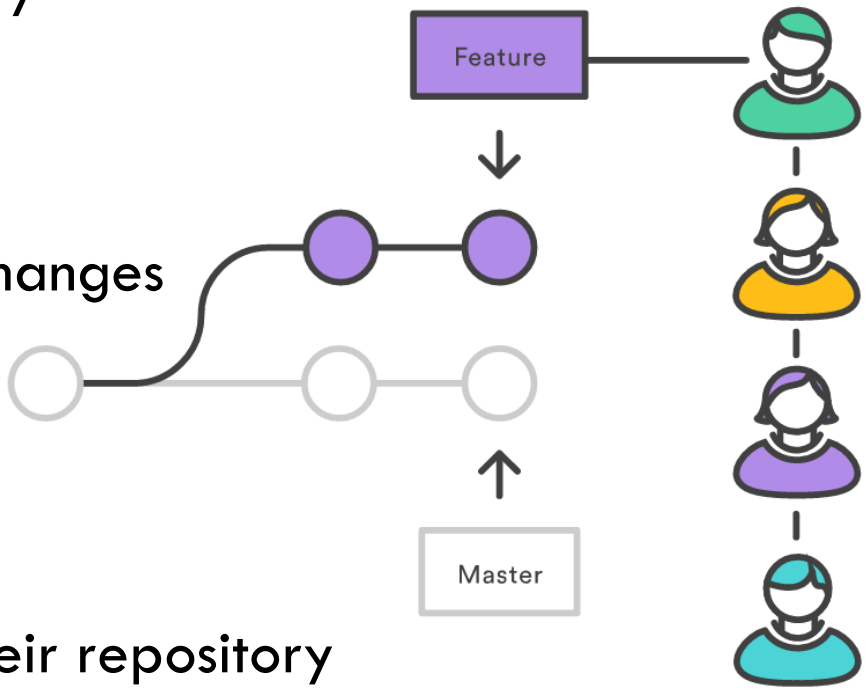


If someone breaks the production branch/trunk in SVN, it blocks everyone else from committing, with GIT you can continue

https://www.atlassian.com/git/

Pull requests

- A pull request is where you ask another developer to merge your feature into their repository
- Proj. leads can keep track of changes
- Proj. leads can merge it with their repository



https://www.atlassian.com/git/

Source control - overview

- Version (Source) control
  - Version control software
  - Why we need it?
- Git
  - What is Git
  - Branching
  - Pull requests

USING GIT AND GITHUB IN OUR APPS – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

Ollscoil na Gaillimhe
University of Galway

Source control - overview

- Git
  - Installing Git
- Adding Git to your projects
  - Committing code
- GitHub
  - Pushing source to remote GitHub repo
  - Cloning a repository
  - Pull requests

Getting GIT – Windows && Mac

- Install it on your machine
- Downloads available at for Windows and Mac
  - https://git-scm.com/download/windows
  - https://git-scm.com/download/mac
- Go with the default install options
- Set details so that every commit is logged correctly…
  - git config --global user.name "Enda Barrett"
  - git config --global user.email "Enda.Barrett@nuigalway.ie"

GIT – Adding Git to your projects

- Once installed navigate to your app directory (myCoolNewApp) on the command line
- Type *git init* from the root of that directory on the command line
  - Creates a repository in this directory
- Execute command *git add* -A
  - Adds all files in the directory to the local repository
- Execute command *git commit*
  - Commits everything to version control

Commit

- When committing you will be asked to put a comment at the top to indicate what changes you made. It's important to be descriptive here so your colleagues can understand the changes made

- Type *i* to begin inserting text

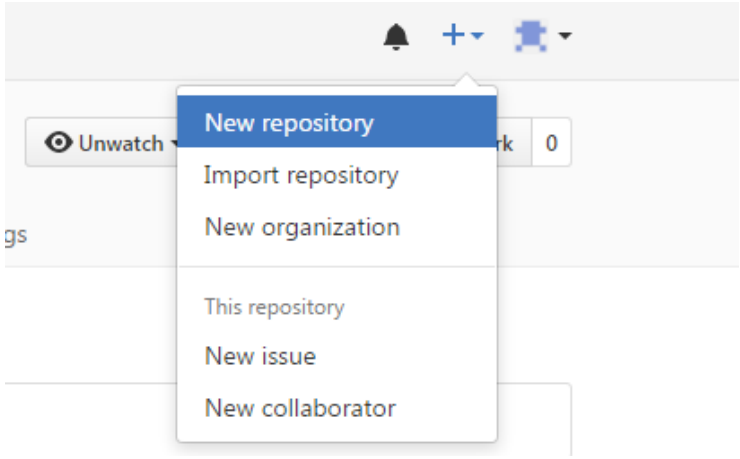
- When finished typing press *Esc* and then *:wq* and hit *Enter*

GitHub

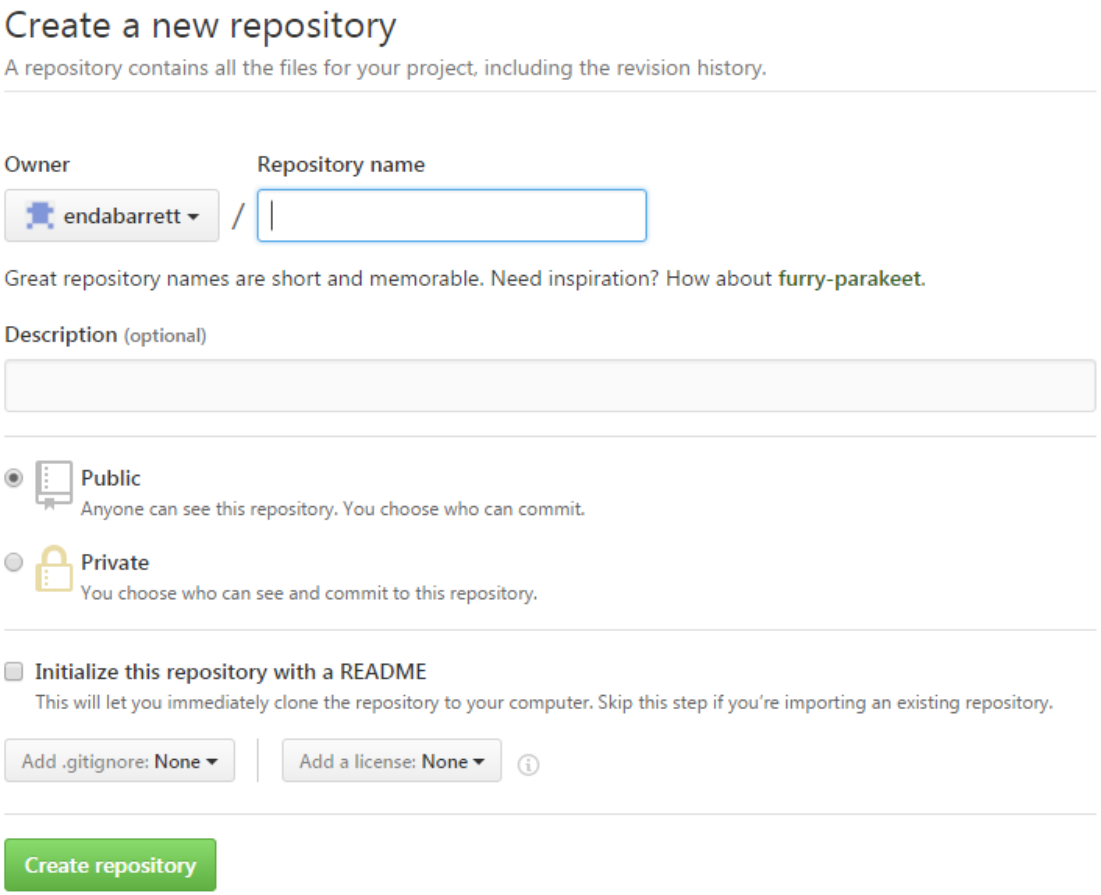
- What is GitHub?
  - GitHub is a web based hosted service for Git repositories. Git allows you to host remote Git repositories and has a wealth of community based services that makes it ideal for open source projects.
- It's really three things
  - A publishing tool
  - Version control system
  - Collaboration platform

Pushing your repository to Github

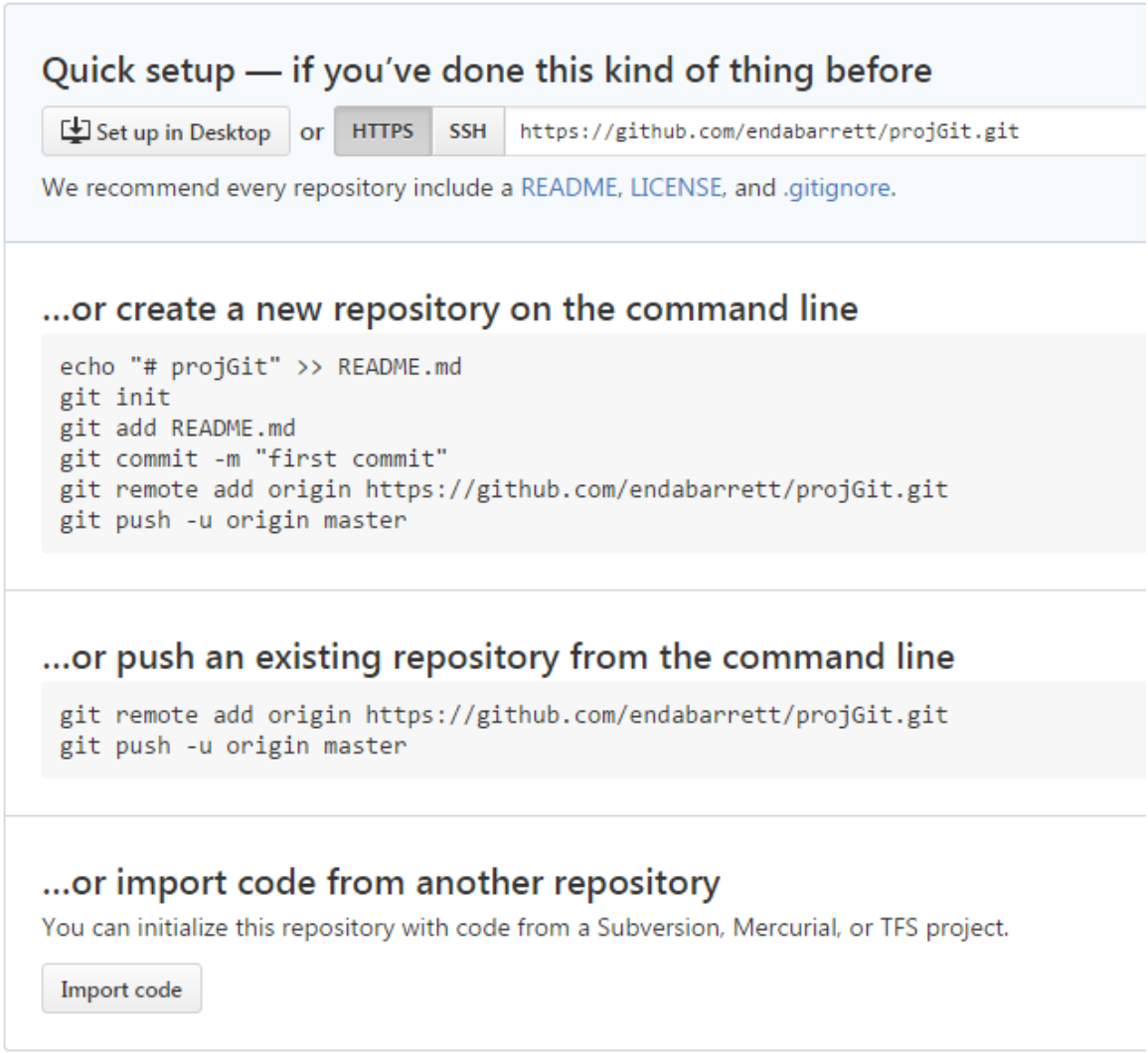
- Create an account on Github
  - It's free for public projects
- Create a new repository



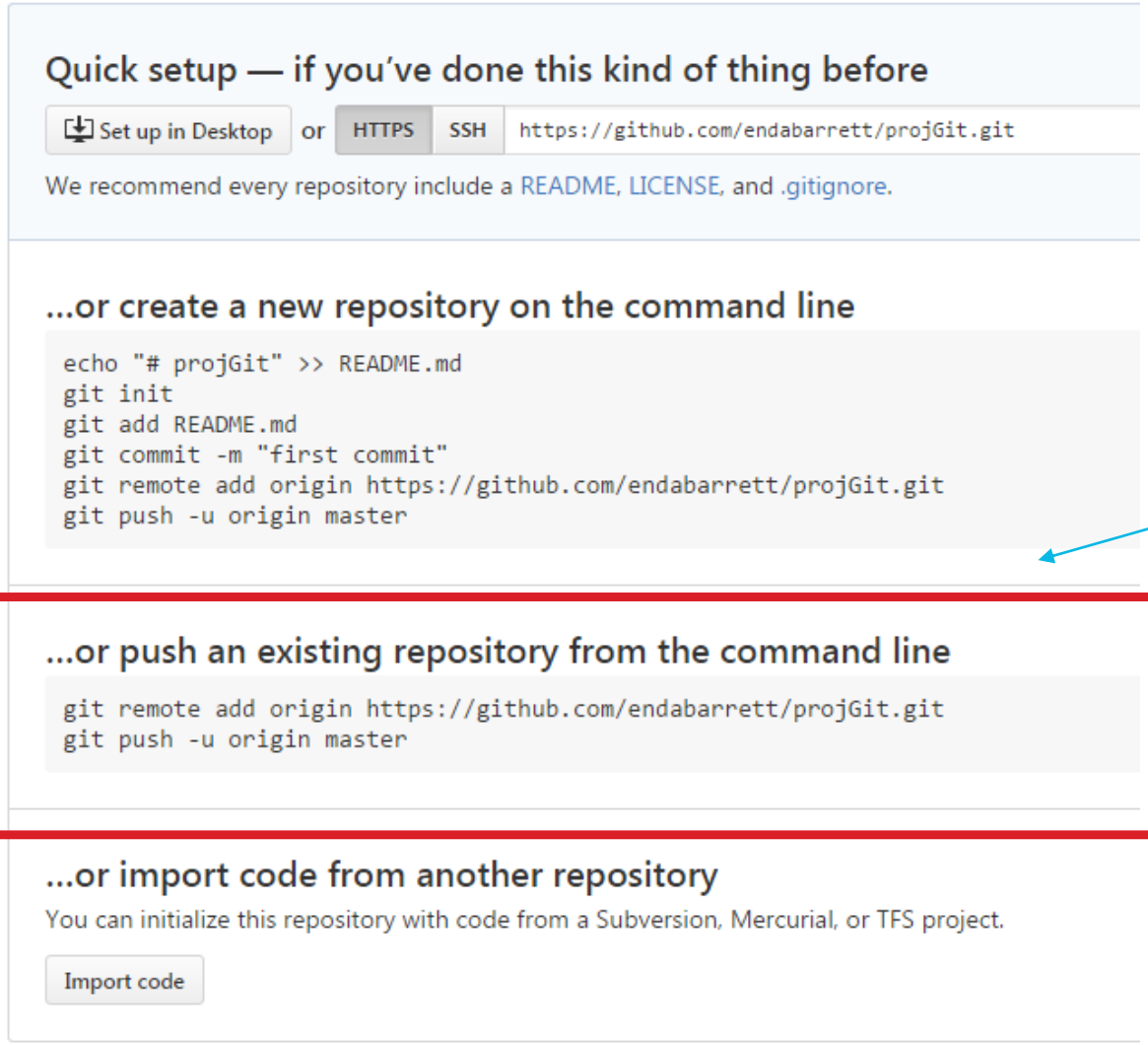
Name it and give it a description



Quick setup options



Pushing to a remote repository



Execute these commands to push to Github

Personal Access Token


```
C:\myCoolNewApp (main)
λ git push -u origin main
Username for 'https://github.com': endabarrett
Password for 'https://endabarrett@github.com':
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
fatal: Authentication failed for 'https://github.com/endabarrett/myCoolNewApp.git/'
```

More info

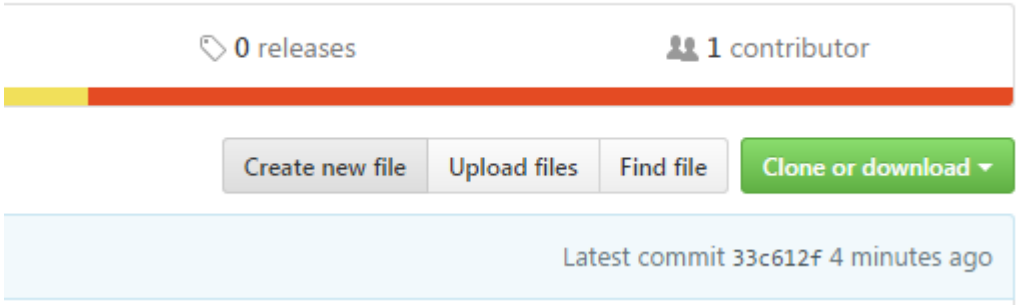
https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/

Steps involved in creating one, please follow these to create the token

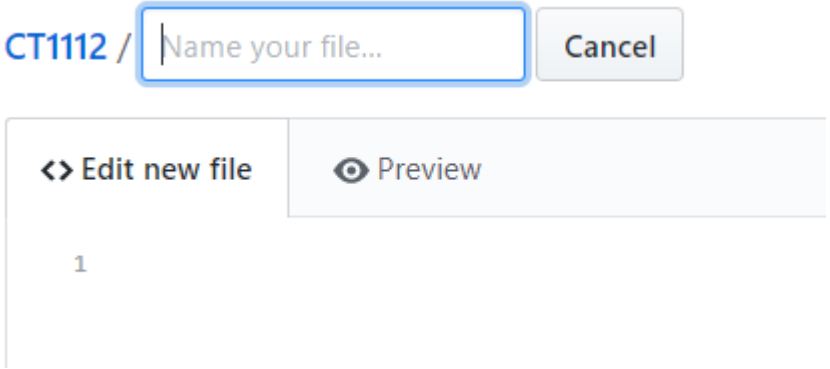
https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Also a files on the web client Github

- Create new file



- Opens up an editor



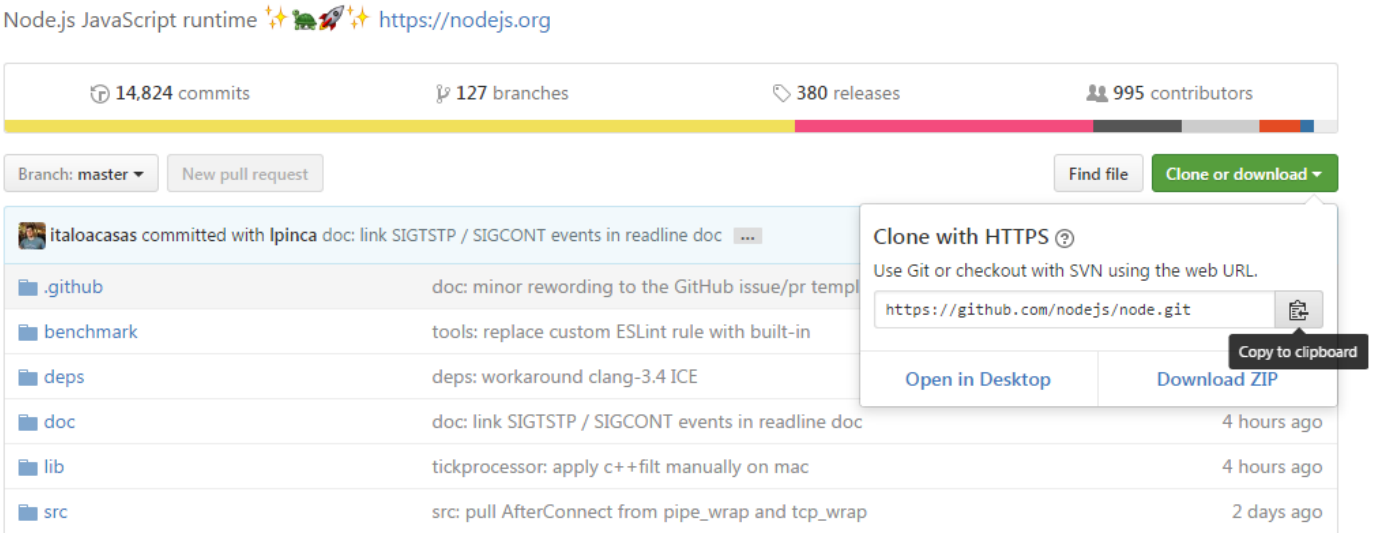
Note: Be careful though, if you are working locally make sure to pull any changes that were made via the web client

Update your repository

- git pull
  - Will pull the latest version from the repository you cloned.
- git fetch and merge
  - Pull is actually a combination of the two and you can run them separately

GIT – Cloning a repository

- Many projects don't require you to create your own repository, instead you clone it from a remote location such as GitHub
- git clone <repo>

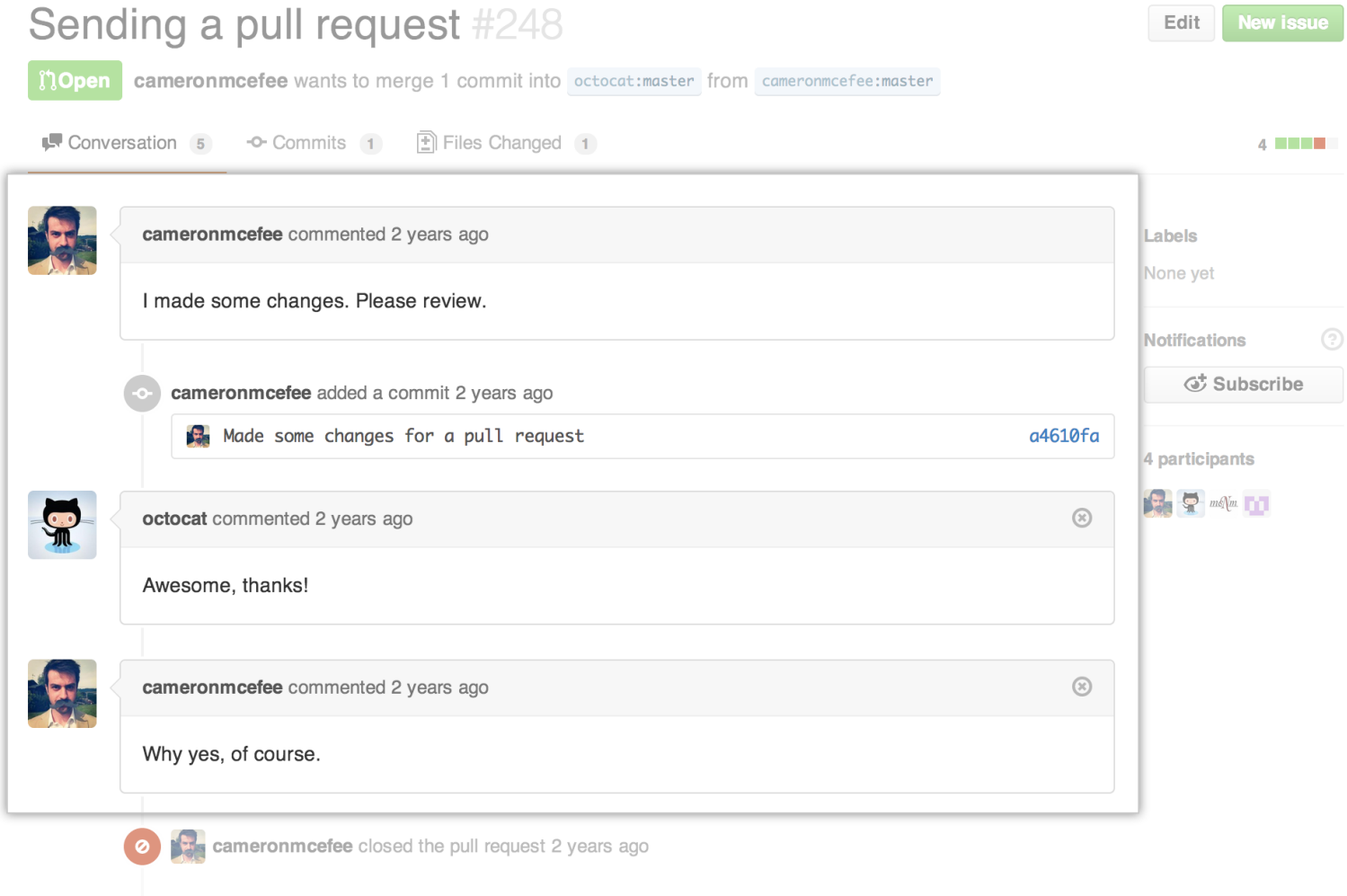


Pull requests

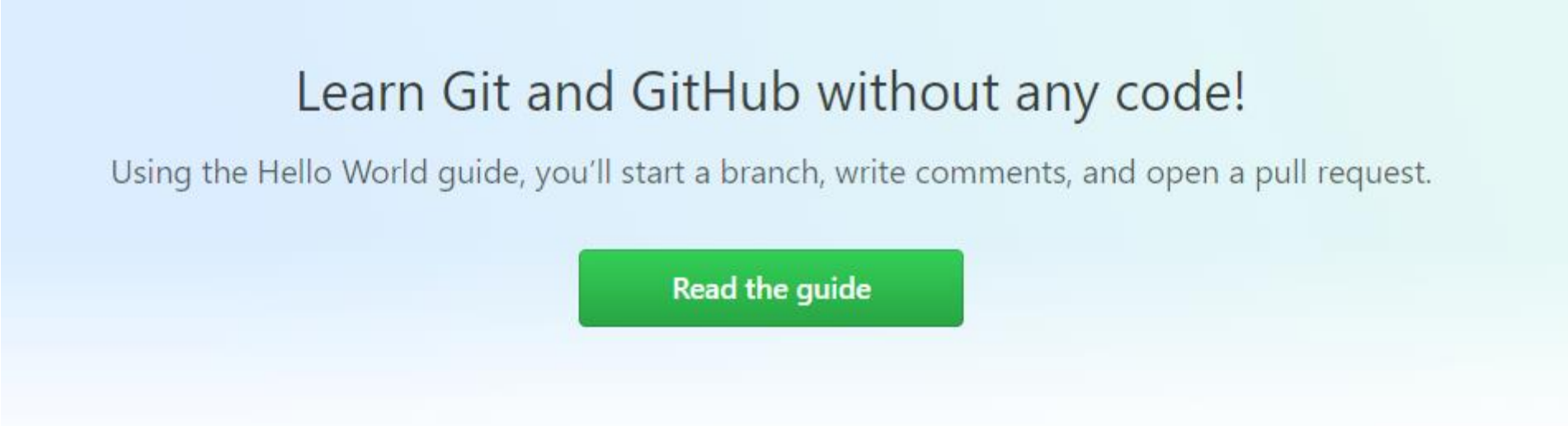
- If you make a change i.e. add a feature you can create a pull request.
- Everyone can review the code and decide whether or not it should be included in the *master* branch
- It's a forum for discussing the changes

- Git commands https://git-scm.com/docs

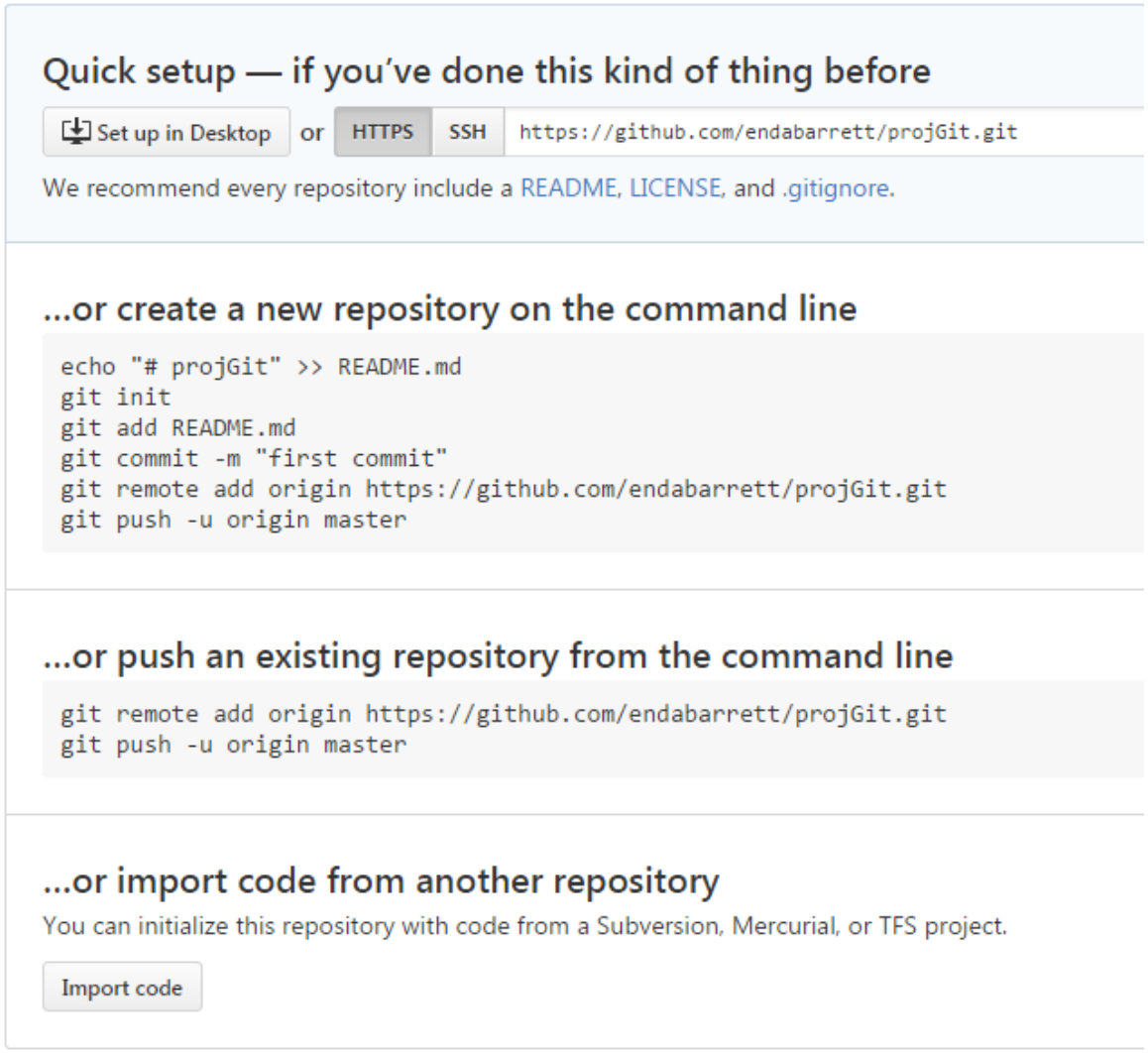
Sample pull request



Excellent Guide



Practical – Creating a repo



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/endabarrett/projGit.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# projGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/endabarrett/projGit.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/endabarrett/projGit.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Commit some changes

- Make some changes to your app's codebase
  - git add –A
  - git commit
  - git push –u origin <master>

Creating branches (locally and remotely)

- To list all branches (local, remote)
  - git branch
  - git branch –r
- Adding a branch locally
  - git branch <branch_name>
- Push it to the remote repo
  - git push -u origin <branch_name>

Switching to a branch and commiting

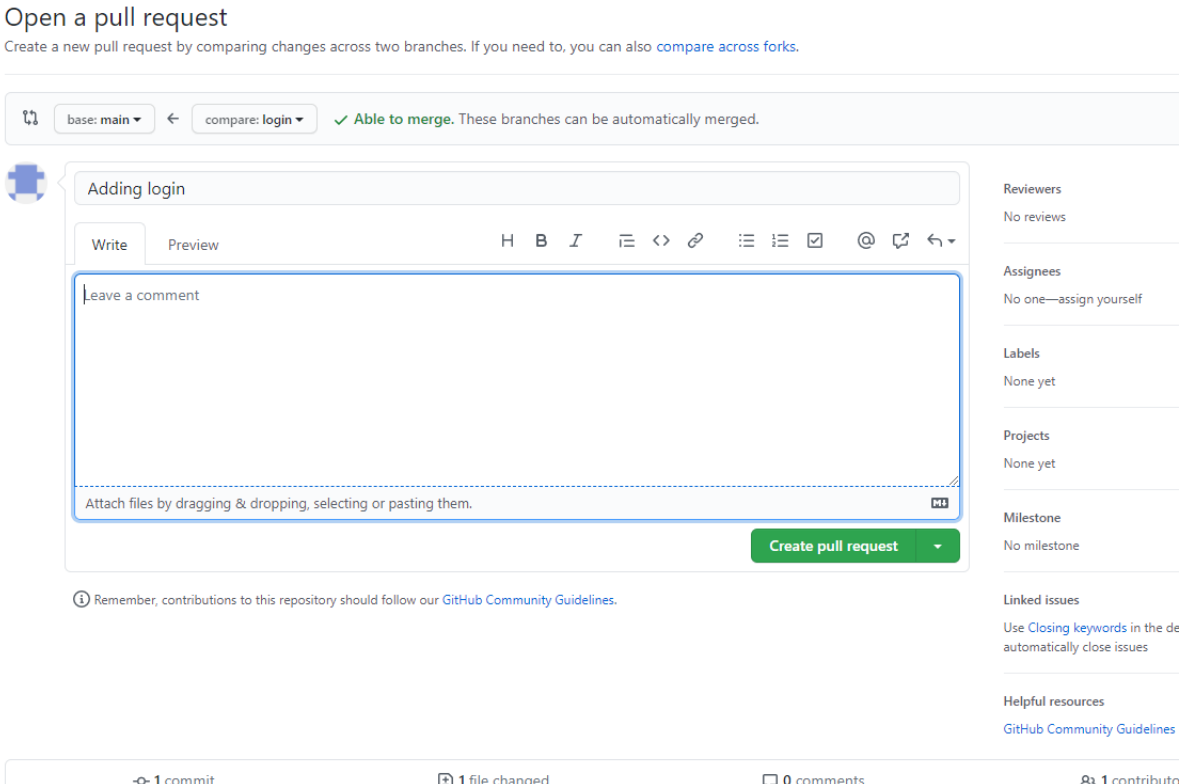
- Now that you've created a branch, we need to check it out so that we can start committing to it
  - git checkout <branch_name>
  - git add –A
  - git commit
  - git push –u origin <branch_name>

If the branch is already remote

- If one of your colleagues has created a branch and you want to work with it
  - git fetch
  - git checkout <branch_name>
  - git add –A
  - git commit
  - git push –u origin <branch_name>

Pull requests

- You've completed your feature and integrate it with the main master branch
- Best to do it using the GitHub web client



Deleting the branch

- Once you have finished and pulled your code into the main master branch, you may want to delete the branch
  - git branch -d <branch_name>
  - git push origin --delete <branch_name>

INTRODUCTION TO VUEJS

Overview

- Introduction to VueJS
  - Background on VueJS
  - Why do we need it?
- How do we started/resources
- Installation guide
- Getting started with Vue
  - Creating a new Vue App
- Examining the main aspects
  - Components
  - Templates
  - Props / Mustache tags

Semester 1 vs Semester 2

- In semester 1 we built a basic database driven web application with a Firebase backend.
  - Semester 1 could be considered the fundamentals of Web Development
- In semester 2 we will now take this basic knowledge of (HTML, CSS, JavaScript and Firebase) and move into a more “advanced” web application.
  - Semester 2 should be considered advanced Web Development

Background on VueJS

- **Vue (pronounced view)** is "what is known as" a progressive framework for building user interfaces.
- In the same way that **Node.js** (Node) runs our back-end code we will use a JS framework called VueJS for our frontend code.
- Good news again! - It's just more JavaScript!
- If you have ever heard of Frameworks such as ReactJS or AngularJS, Vue is the same

Why do we need it?

- Couldn't we just continue with our apps from last semester?

- The short answer is, **yes**. These frameworks whilst ubiquitous in industry at the moment, are not mandatory for building web applications.

- However, they will speed up development significantly, enforce proper practices and good standards. They will also enable you to **avoid** "boatloads" of JS code that could prove difficult to maintain as the project progresses.

How do we get started/resources?

- The VueJS web page is an excellent resource and I'll be using many of their examples throughout the upcoming weeks
  - https://vuejs.org/
- There are ample books on the subject. I haven't chosen a specific text because I don't want to bind the learning to a specific author's structure.
- However, some of the most recommended texts are
  - Fullstack Vue by Hassan Djirdeh
  - Vue.js 3 By Example by John Au-Jeung

Installation

- There are multiple ways of installing this (4 primary ways).
- See details here
  - https://vuejs.org/guide/quick-start.html

1. Import it as a CDN package on the page
2. Download the JavaScript files and host them yourself
3. Install it using npm
4. Use the official CLI to scaffold a project, which provides batteries-included build setups for a modern frontend workflow (e.g., hot-reload, lint-on-save, and much more)

Create our first project

- Create a new directory called vueapps
- This is where we will create our vue apps.
- Again it's helpful to place these on the root of your machine, for easy command line access.

Option 4 – Installing Vue

- Information about it
  - https://vuejs.org/guide/quick-start.html#creating-a-vue-application
- Open up a command prompt and type the command within the newly created vueapps folder
  - npm init vue@latest
- This installs and executes *create-vue* the scaffolding tool for vue.

Creating a project

- The scaffolding tool allows you to create a base project with various options.
- You can set the project features and tools to integrate as part of the project

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add Cypress for both Unit and End-to-End testing? … No / Yes
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

Creating a project cont.

- For our first “Hello World” type project, just use the default options

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add Cypress for both Unit and End-to-End testing? … No / Yes
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

- Note – Don’t do this in industry – in an academic environment Linting will create too many formatting and prog. errors amongst the group that will drain a huge amount of time. Many of these are minor and don’t affect the functionality.
- The other components, i.e. Router, Pinia (Vuex) etc will all be added later

The new app should now be ready

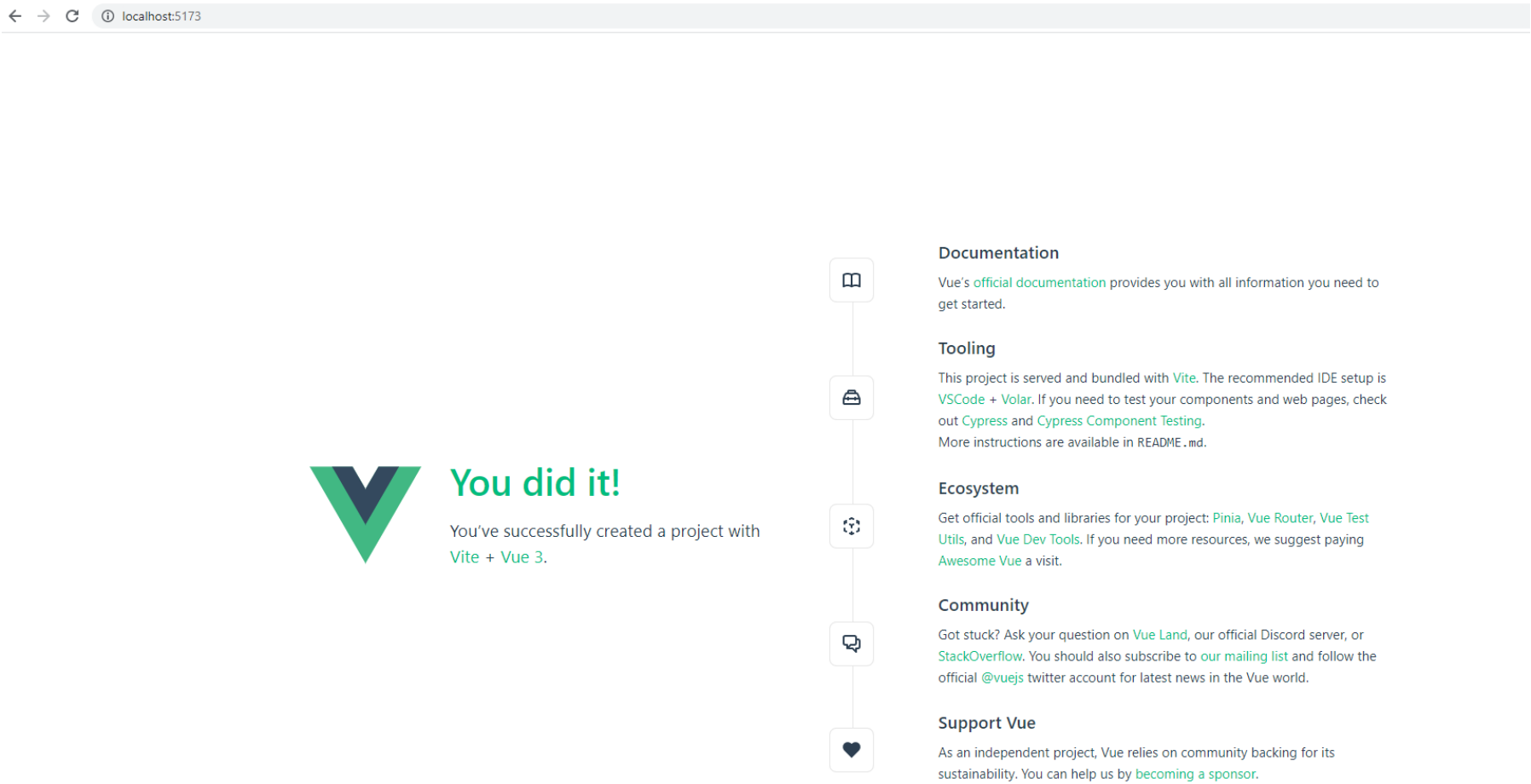
- Change directory into the new app

```
Done. Now run:

  cd myProject
  npm install
  npm run dev
```

- Type npm run dev

```
  VITE v4.0.4  ready in 505 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

Goto – http://localhost:5173



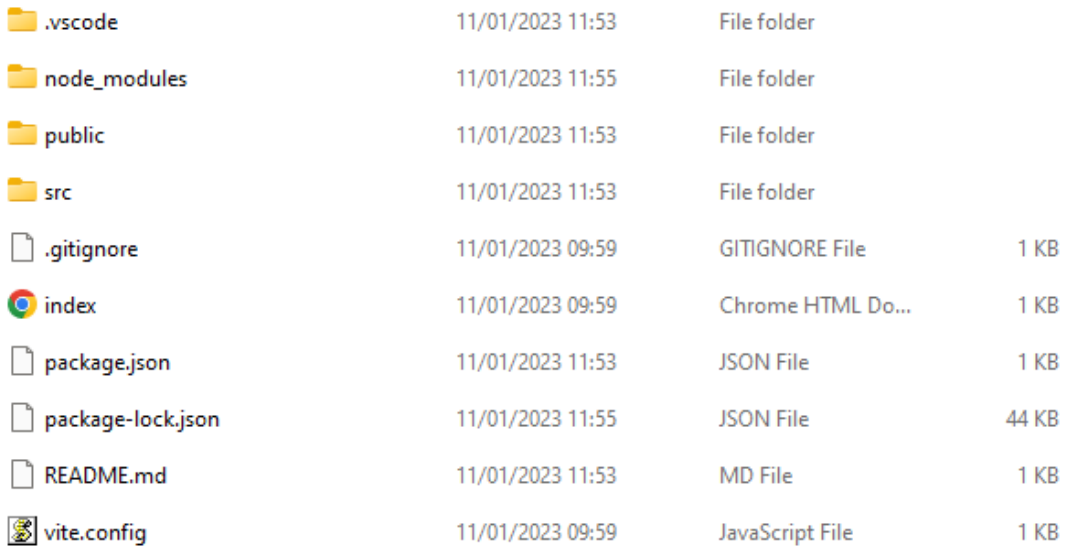
Exercise 1 – Installation and Creation

- Install Vue on your local machines via the command line – see slide 8 - 13

- Run it and test it works

Folder structure – hello-world app

- A bunch of config files and some folders, some of which look familiar.
- The src folder is where we will build out our Vue apps



.vscode	11/01/2023 11:53	File folder	
node_modules	11/01/2023 11:55	File folder	
public	11/01/2023 11:53	File folder	
src	11/01/2023 11:53	File folder	
.gitignore	11/01/2023 09:59	GITIGNORE File	1 KB
index	11/01/2023 09:59	Chrome HTML Do...	1 KB
package.json	11/01/2023 11:53	JSON File	1 KB
package-lock.json	11/01/2023 11:55	JSON File	44 KB
README.md	11/01/2023 11:53	MD File	1 KB
vite.config	11/01/2023 09:59	JavaScript File	1 KB

Public folder

- Last semester we spent much of our time developing our content within the *public* folder.

- This time there is only a favicon in there by default.

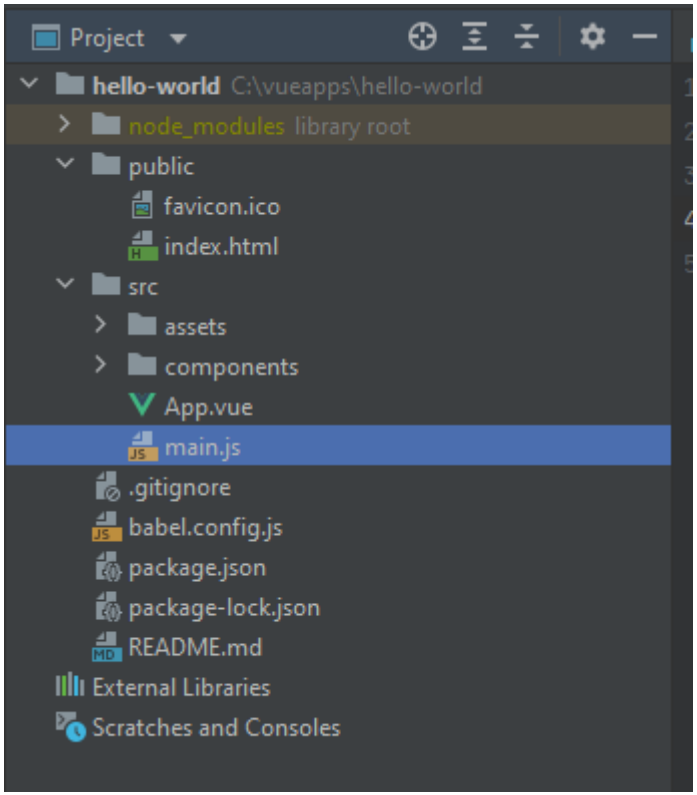
index.html

- The index page provides the entry point in HTML.
- It provides an element for VueJS to load into

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vite App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

src folder

- This folder contains the src vue code for your app



Main.js - explained

- The JS file that initializes the root component into an element on your page.
- The element is “app” see it in index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vite App</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

Main.js cont.

- Imports the Vue object “createApp” from the library (located in the node modules folder – note no path needed)

```
import { createApp } from 'vue'
import App from './App.vue'

import './assets/main.css'

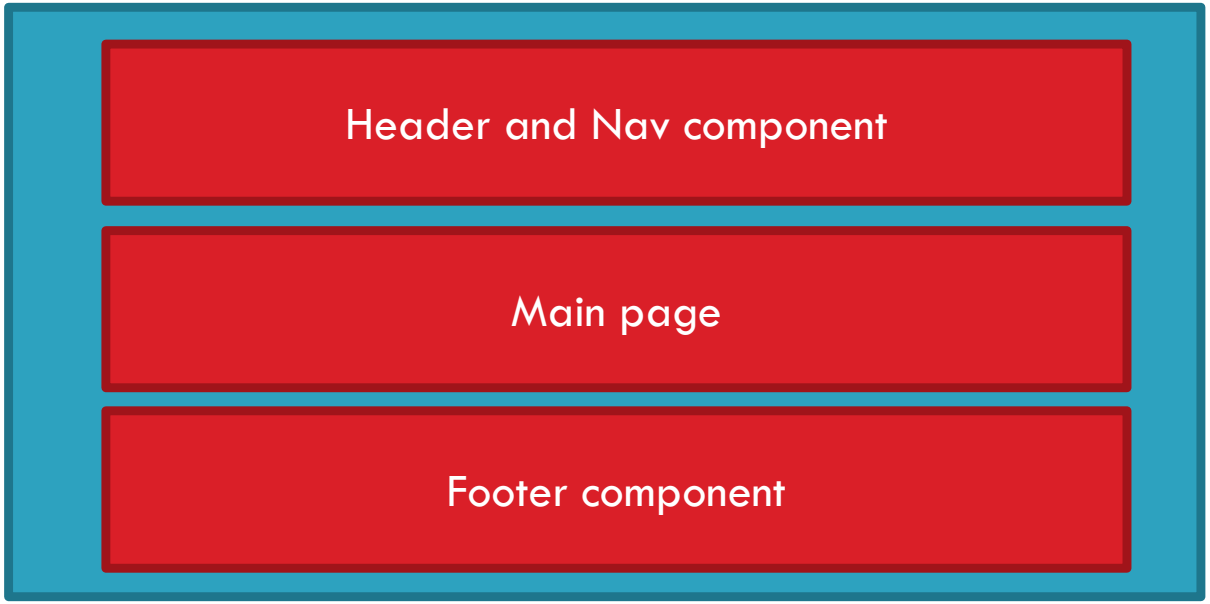
createApp(App).mount( rootContainer: '#app')
```

- Imports the App.vue file from the current directory and passes it in as an argument to vue – specifying the ID of the HTML tag that hosts the App.vue component

https://vuejs.org/guide/essentials/application.html

Components – Single File Components

- Any file you see with a .vue extension is a Single File Component
- The idea is that you build reusable components for different sections/segments of your web page which you can plugin and reuse as often and wherever you need.



Three primary parts to a component


```
<script setup>
defineProps( props: {
  msg: {
    type: String,
    required: true
  }
})
</script>

<template>
  <div class="greetings">
    <h1 class="green">{{ msg }}</h1>
    <h3>
      You've successfully created a project with
      <a href="https://vitejs.dev/" target="_blank" rel="noopener">Vite</a> +
      <a href="https://vuejs.org/" target="_blank" rel="noopener">Vue 3</a>.
    </h3>
  </div>
</template>

<style scoped>
h1 {
  font-weight: 500;
  font-size: 2.6rem;
  top: -10px;
}

h3 {
  font-size: 1.2rem;
}

.greetings h1,
.greetings h3 {
  text-align: center;
}

@media (min-width: 1024px) {
  .greetings h1,
  .greetings h3 {
    text-align: left;
  }
}
</style>
```

JavaScript

HTML - template

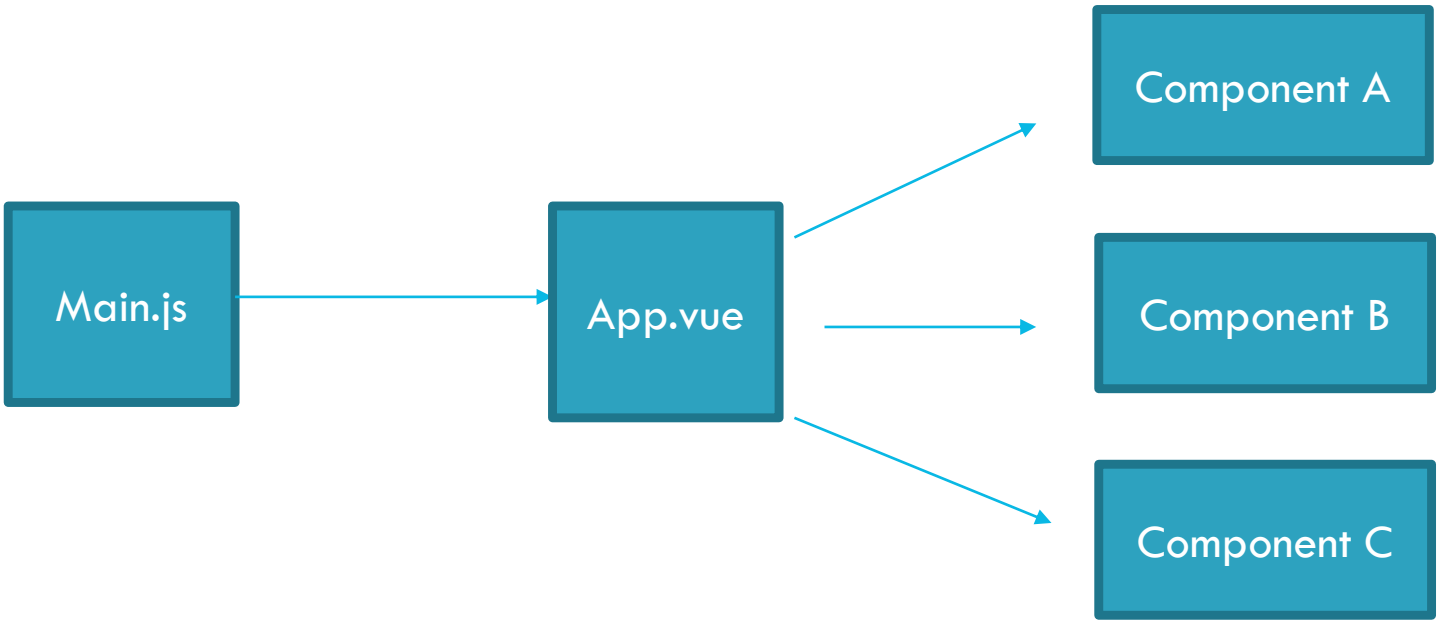
CSS

Components are encapsulated

- This makes components encapsulated, where the HTML, JS and CSS for a specific component is all neatly present within a single file. The CSS can be scoped to a component.
- You can reuse components throughout the app wherever you wish.
- They need to be pre-compiled by a vue compiler, the browser won't recognize .vue files and know implicitly what to do with them.
- It compiles into a standard ES module

App.vue

- This is the entry point or root level component. Use this to import other components into your application



Templates

- Template tags are very useful as you can place content within them that will remain hidden when the page loads.
- Using JavaScript it can be displayed.
- Within the opening and closing template tag you place the HTML you would like to render for the component.
- Using the example of the hello-world vue app we have just created. If you look under src>components>HelloWorld.vue you will see lots of HTML, that forms the content when you open localhost:5173
- The vue engine will dynamically load the templates required for each page

CSS styles - component

- Under the style tag usually at the bottom of the Vue SFC files you can specify styles for the component.
- Styles can be scoped so that the styles only apply to a specific component

```
<style scoped>
h3 {
  margin: 40px 0 0;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  display: inline-block;
  margin: 0 10px;
}
a {
  color: #42b983;
}
</style>
```

JavaScript – vue SFC

- Within the <script> tags you place the JS.

```
<script setup>
defineProps({
  msg: {
    type: String,
    required: true
  }
})
</script>
```

Defining properties on the page that are passed in as args

```
<template>
  <div class="greetings">
    <h1 class="green">{{ msg }}</h1>
    <h3>
      You've successfully created a project with
      <a href="https://vitejs.dev/" target="_blank" rel="noopener">Vite</a> +
      <a href="https://vuejs.org/" target="_blank" rel="noopener">Vue 3</a>.
    </h3>
  </div>
</template>
```

HelloWorld.vue

```
<div class="wrapper">
  <HelloWorld msg="You did it!" />
</div>
```

App.vue

Including an imported component


```
<script setup>
import HelloWorld from './components/HelloWorld.vue'
import TheWelcome from './components/TheWelcome.vue'
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="./assets/logo.svg"
width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="You did it!" />
    </div>
  </header>

  <main>
    <TheWelcome />
  </main>
</template>
```

Imported components must be included in the template by surrounding them in angle brackets. Any “props” can be treated like tag attributes

When compiling, Vue will search for any components and listed under the property “components” and render them

Props

- You can pass data between components using props.
- In the Hello World example we can pass data between components using these props.
- App.vue

```
<HelloWorld msg="You did it!" />
```

- HelloWorld.vue

```
<script setup>
defineProps({
  msg: {
    type: String,
    required: true
  }
})
</script>
```

Mustache tags – data binding

- Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying component instance's data.
- Using the mustache templating system we can bind props to mustache tags {{ msg }}
- https://mustache.github.io/

```
<h1 class="green">{{ msg }}</h1>
```

https://vuejs.org/guide/essentials/template-syntax.html

```
<script setup>
defineProps({
  msg: {
    type: String,
    required: true
  }
})
</script>
```

Hot reload

- Once you have the app running you can view changes in real time as you make them in the editor.
- This is known as hot reloading
- Test this for yourself by modifying the HelloWorld.vue component in real time

Exercise 2 – Adding a component

- Create a new component called footer
- Create a list of empty links – about, home etc.
- Import the new footer component into the main App.vue component and display it on the page

VUEJS, ADDING FIREBASE AND ROUTING

OÉ Gaillimh
NUI Galway

Overview

- Ongoing issues/observations with the framework

- Adding Firebase Hosting to our Vue App
- Adding Firebase Functions to our Vue App

- Routing
  - Installing Vue Router
  - Client side routing
  - Creating routes
  - Router links

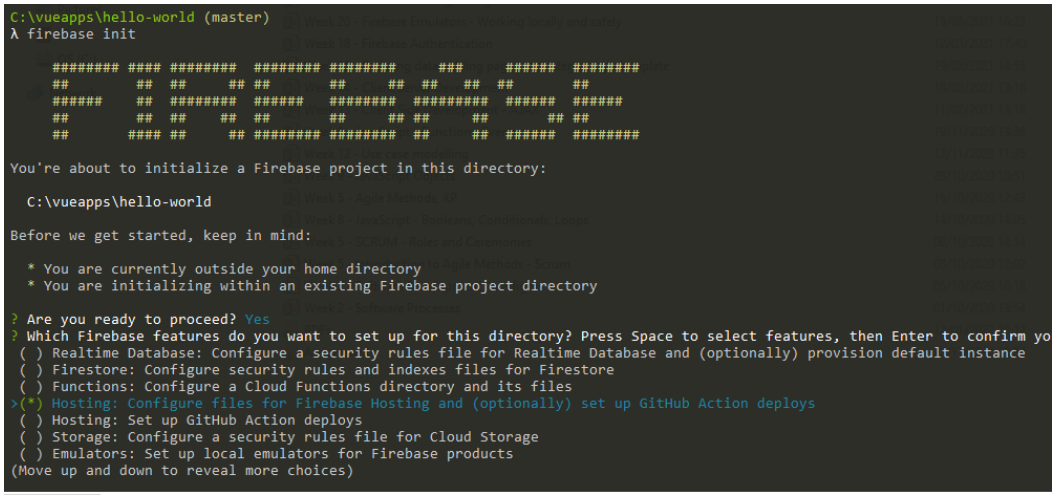
1. SFCs must start with a capital

- When naming Vue components (Single File Components) they must start with a capital letter otherwise they won't compile properly and will given an error.

- You don't need to specify any JS in the SFC and it will still work (seems to be rather flexible)

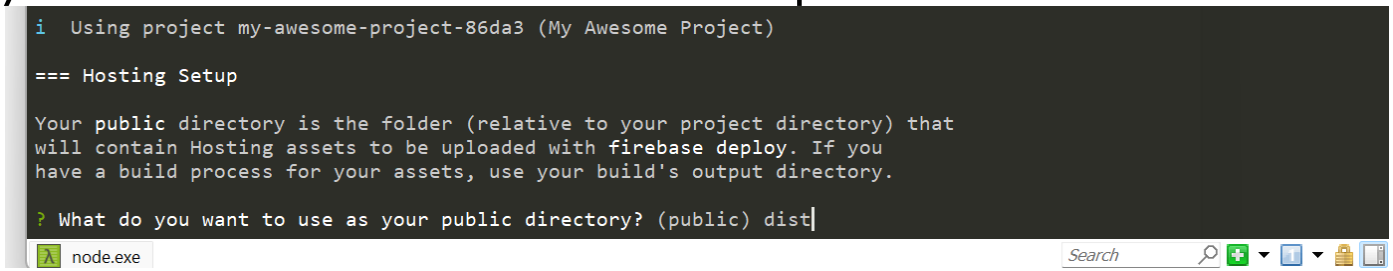
Adding Firebase to our App

- Good News, it's just a matter of running the ***firebase init*** command and associating a project with it.
- Open CMD/Terminal
- Navigate to the vueapps/myApp folder
  - Execute the command firebase init
- Let's start with hosting by selecting it and hitting spacebar



Adding Firebase hosting

- Since we already have a project from last semester with some functions already written let's associate the app with this (my-awesome-project…).
- Specify a **dist** folder instead of the default public



- Select Yes to configure a single-page app
- Select No for automatic builds and deploys with GitHub
- Select No to overwriting index.html

```
? What do you want to use as your public directory? public
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
? Set up automatic builds and deploys with GitHub? No
? File public/index.html already exists. Overwrite? No
i  Skipping write of public/index.html

i  Writing configuration info to firebase.json...
i  Writing project information to .firebaserc...

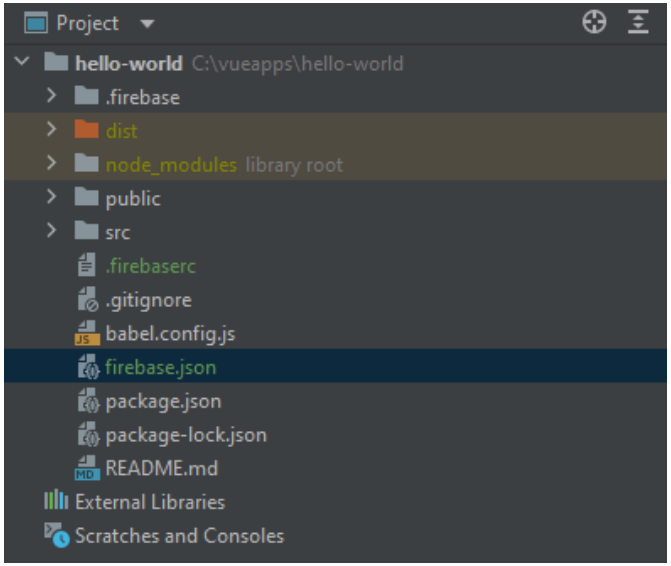
+  Firebase initialization complete!
```

Generating a build for deployment

- The command npm run dev creates a local server on the machine and deploys the application to it.
- This is fine for development, but not for production.
- In order to deploy to a cloud backend like Firebase we need to bundle up the application so that all the files, assets, CSS etc. are in a suitable format for deployment.
  - This is known as generating a **build**
- Execute the command npm run build
  - Once the command has finished the build will be placed in the **dist** folder

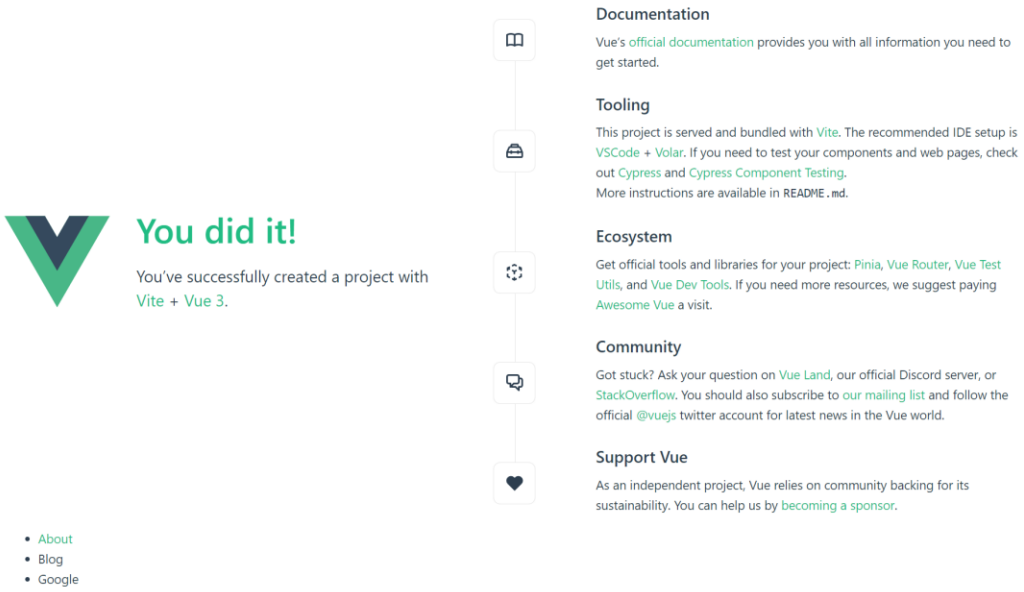
dist folder

- VueJS packages up the application and places into the dist folder – short for distribution
- None of the common errors, warnings will be included in the production build. As they bloat the application payload size.



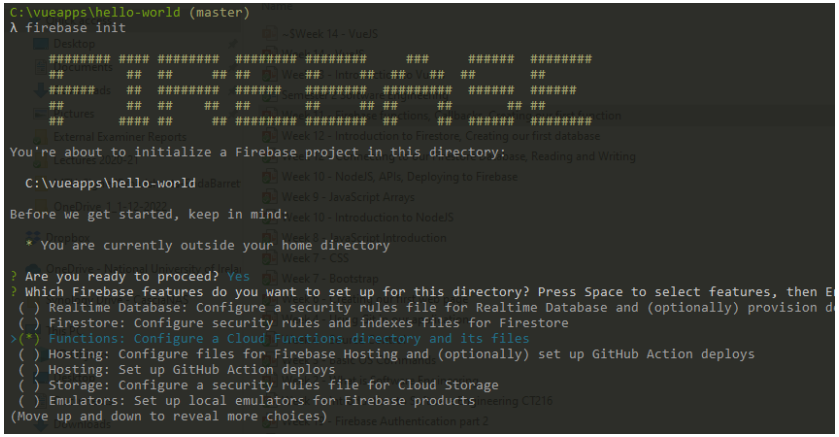
Firebase deploy

- Run the command *firebase deploy* and your app should now be deployed onto Firebase



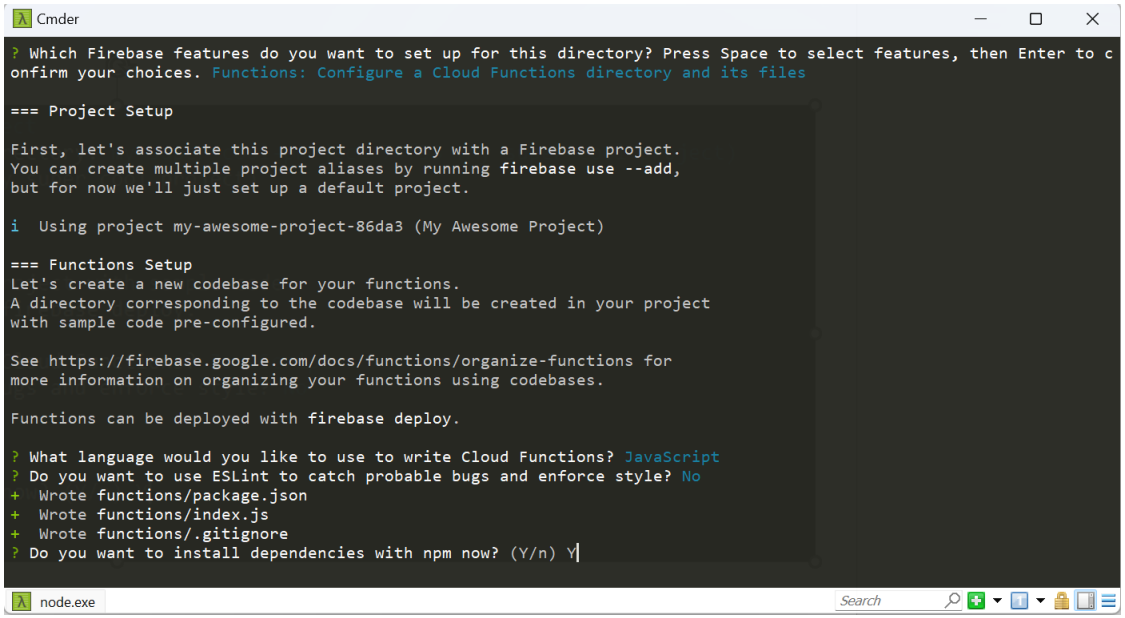
Adding Firebase Functions to our App

- Again, it's just a matter of executing the *Firebase Init* command!
- Open CMD
- Navigate to the vueapps/myProject folder
  - Execute the command firebase init
- Add functions by selecting it and hitting spacebar



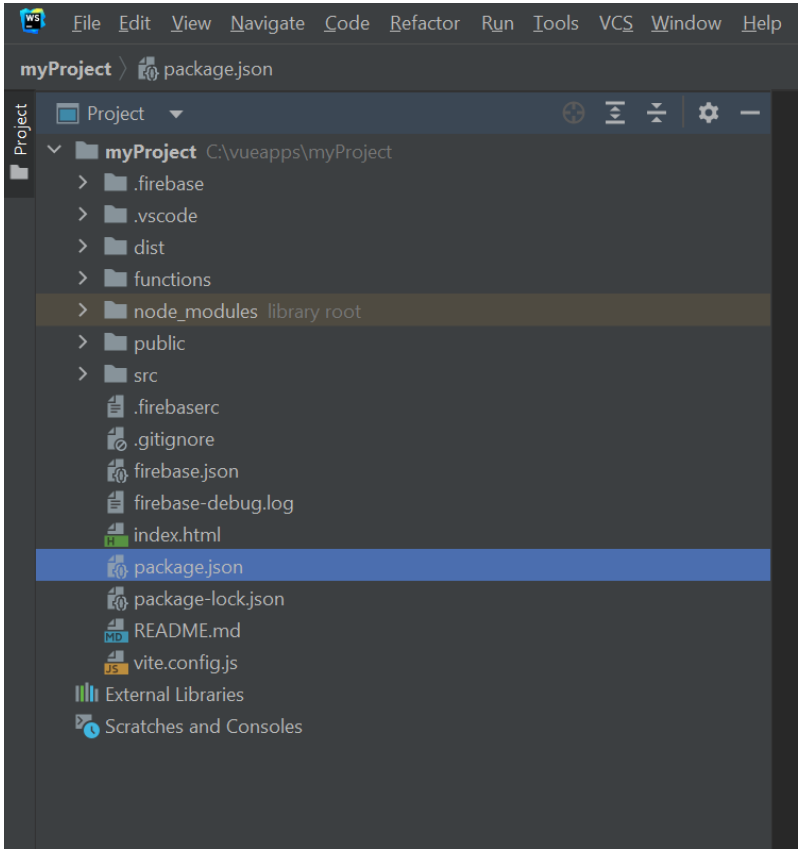
Adding Firebase cont.

- Select JavaScript
- Select no for linting
- Install dependencies



Functions folder

- If it has worked correctly, you should now see a functions folder



Exercise 1 - Deployment

- Add both Firebase Hosting and Firebase Functions to your new Vue App

- Copy your existing functions index.js (from the older project – if you still have it) into the new folder
  - This file contains your functions for getComments and postComments (see week 11)

- Deploy the app and Firebase should now be hosting the new Vue Application

Server-side routing

- Routing on the server is where a client receives a response based on the URL path that the user is visiting.
- In a traditional web app setup (what we built last semester), when clients requested our pages, index.html, about.html etc. the server responded to each request with HTML, CSS and JS for each page and reloaded the entire page with the new content

Single Page App

- Client side JavaScript, intercepts the navigation (/index, /about, /login etc), requests the data from the server and updates the page without a full page reload.
- This typically results in a much faster user experience, as many pages can be constructed and loaded in memory without having to be requested from the server at all.
- In SPAs “routing” is done on the client to load the various web pages.
- In VueJS Single File Components are loaded based on the path selected by the user

Routing

- With a Single Page App there is only one page on the server – but it is useful to support address bar route navigation i.e. /about-us or /login for bookmarking and usability etc.
- A client-side router can achieve all of this by parsing the requests on the client and displaying the necessary content each time.
- Docs *https://router.vuejs.org/*

Routing cont.

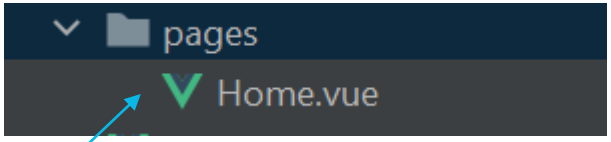
- We can use the vue-router package written for VueJS to enable client-side routing for specific paths
- To the end user the application will look like a normal multi-page server-side app, but in reality, it will be just simply be displaying SFC components based on the path entered.
- You will notice that it is incredibly responsive and fast, this is because the application is not going back to the server for each request.

Housekeeping – pages and components

- Having built web apps for many years I still like to work under the convention of there being pages.
- I understand that with Single Page Apps this doesn't technically exist, but I think it's an easier convention to follow.
- I personally prefer to structure my app to have a folder called *pages* (containing SFCs that represent a logical page) and a folder called *components* (containing SFCs that are shared amongst pages).

Add a “pages” directory

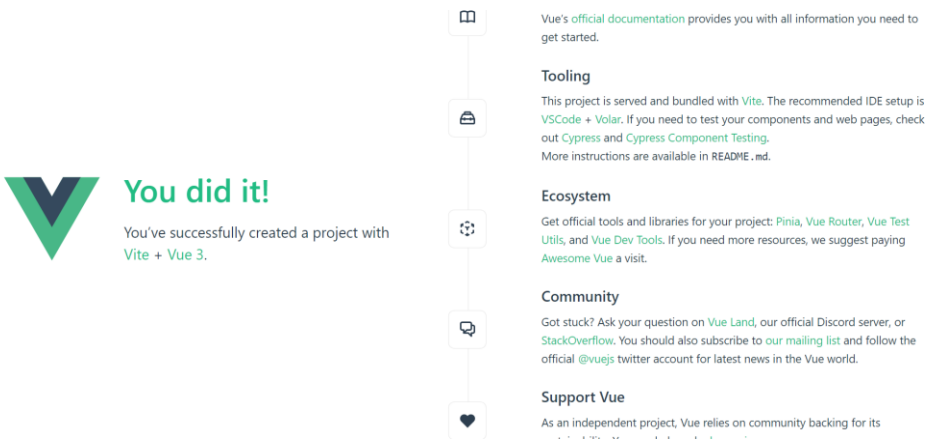
- We’re going to restructure the app into pages. The first thing is to create a Home.vue page.



- Add a SFC to the pages folder call **Home.vue**

Home page

- Currently there are a number of vue components which are imported by the main vue component App.vue and the result is the page below
- This is available at the root URL "/"
- Let's now import them into our new Home page



Edit Home.vue

- Edit the newly created Home.vue SFC and add the following

```
<script setup>
import HelloWorld from '../components/HelloWorld.vue'
import TheWelcome from '../components/TheWelcome.vue'
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="../assets/logo.svg" width="125" height="125" />

    <div class="wrapper">
    <HelloWorld msg="You did it!" />
  </div>
  </header>

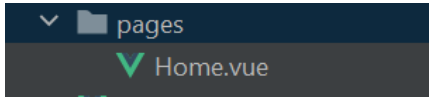
  <main>
    <TheWelcome />
  </main>

</template>

<style scoped>

</style>
```

Much of this is simply removed from the current App.vue file



Delete existing content from App.vue

- Modify App.vue to import the new “Home” page and test that it works.

App.vue

```
<script setup>
import Home from './pages/Home.vue'
</script>

<template>
  <Home />
</template>

<style scoped>
header {
  line-height: 1.5;
}

.logo {
  display: block;
  margin: 0 auto 2rem;
}

@media (min-width: 1024px) {
  header {
    display: flex;
    place-items: center;
    padding-right: calc(var(--section-gap) / 2);
  }

  .logo {
    margin: 0 2rem 0 0;
  }

  header .wrapper {
    display: flex;
    place-items: flex-start;
    flex-wrap: wrap;
  }
}
</style>
```

Home page

You did it!

You've successfully created a project with Vite + Vue 3.

Documentation

Vue's official documentation provides you with all information you need to get started.

Tooling

This project is served and bundled with Vite. The recommended IDE setup is VSCode + Volar. If you need to test your components and web pages, check out Cypress and Cypress Component Testing.
More instructions are available in `README.md`.

Ecosystem

Get official tools and libraries for your project: Pinia, Vue Router, Vue Test Utils, and Vue Dev Tools. If you need more resources, we suggest paying Awesome Vue a visit.

Community

Got stuck? Ask your question on Vue Land, our official Discord server, or StackOverflow. You should also subscribe to our mailing list and follow the official @vuejs twitter account for latest news in the Vue world.

Support Vue

As an independent project, Vue relies on community backing for its sustainability. You can help us by becoming a sponsor.

Adding additional pages

- We now have a single “home” page but what if we wish to add additional pages?
- This is where routing comes in, we need to load the different “pages” (which are themselves still SFCs) depending on the path.
- We can use a client-side router within Vue to do this for us.

Install vue-router

- Open up cmd and install vue-router using the following command
  - npm install --save vue-router@latest
- This will install the latest version of vue-router (v4) via NPM. It will pop it into the node_modules folder and add an entry to package.json for deployment

Package.json

Router-view

- We wish to display whatever component is requested by the user, based on the inputted path.
- The <router-view> custom component will enable this.
- Pop it into App.vue and remove Home.vue as shown

App.vue

```
<script setup>
</script>

<template>
  <router-view></router-view>
</template>

<style scoped>
header {
  line-height: 1.5;
}

.logo {
  display: block;
  margin: 0 auto 2rem;
}

@media (min-width: 1024px) {
  header {
    display: flex;
    place-items: center;
    padding-right: calc(var(--section-gap) / 2);
  }

  .logo {
    margin: 0 2rem 0 0;
  }

  header .wrapper {
    display: flex;
    place-items: flex-start;
    flex-wrap: wrap;
  }
}
</style>
```

Edit main.js

- Import vue-router to use in our apps
- This will configure it to use routes defined in the file routes.js
- Let's now add some routes

```
import { createApp } from 'vue'
import App from './App.vue'
import { createRouter, createWebHistory }
from 'vue-router';
import routes from './router/routes';
import './assets/main.css'

let router = createRouter({
    history: createWebHistory(),
    routes: routes
});

const app = createApp(App)
app.use(router);
app.mount('#app');
```

Main.js

Imports explained

- Default import
- JS supports what's known as a *default export* from a JS file
  - This could be a number, a function, an object, an array…
  - When importing the *default export*

    File x

    ```
    export default 53
    ```

    File y

    ```
    import x from './x';
    ```

  - It doesn't matter what name you assign to it when importing it – when importing the default export

Imports explained cont.

- **Named import**
- JS also supports *named exports* from a JS file
  - This could be a number, a function, an object, an array…
  - When importing *a named export*

    File x

    ```
    export const num = 53
    ```

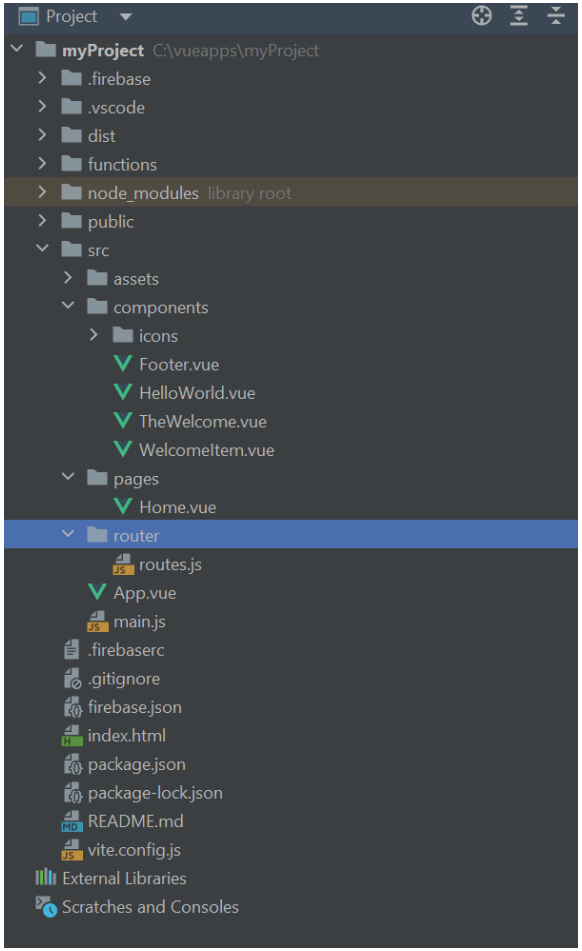
    File y

    ```
    import {num} from './x';
    ```

  - A file may have many named exports – thus you have to specify exactly the one you require
  - It can only have one default export though!

Routes.js

- To define routes, we must create a file called routes.js
- Create a new directory called router to house this file
- Your app should now start to resemble this structure



Routes.js

- The function “loadPage” is a helper function to load components from the “pages” folder.
- This function “loadPage” just imports the named component at runtime.
- The file exports an array of JS objects, which contains the path as well as the component to load.

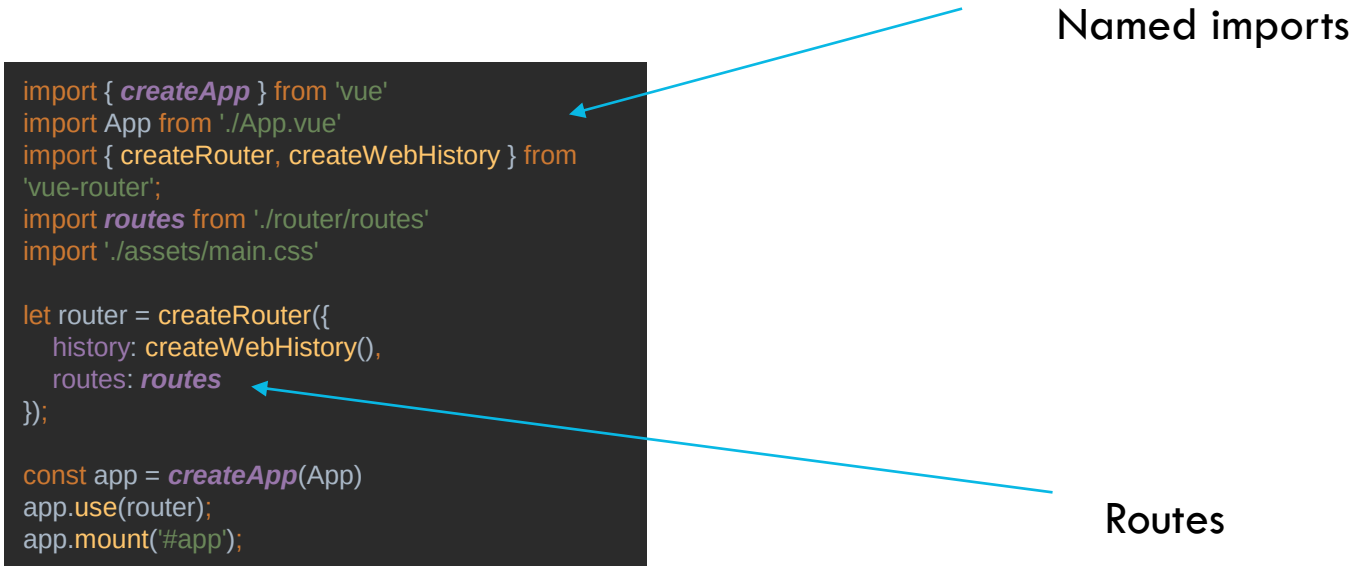
```
function loadPage (component) {
    // '@' is aliased to src/components
    return () => import(/* webpackChunkName: "[request]" */
`@/pages/${component}.vue`)}
export default [
    { path: '/', component: loadPage('Home') }
]
```

String literal specifying the path to the page

Actual page to render when the path is hit

The routes are passed in during init

- In main.js, you can see that the routes are imported and passed into the the vue-router



Main.js

Add a second page “Blog.vue”

- Let’s now create a second page called Blog.vue
- Add it to the pages directory
- Let’s keep it basic for the time being with no imported components

```
<template>
<h1>Welcome to my new Blog page</h1>
  <p>This is my very first blog entry</p>
</template>


<style scoped>

</style>
```

Blog.vue

Add an entry in router/routes.js

- Add an additional route to the new page

```
function loadPage (component) {
    // '@' is aliased to src/components
    return () => import(/* webpackChunkName: "[request]" */
`@/pages/${component}.vue`)}

export default [
    { path: '/', component: loadPage('Home') },
    { path: '/blog', component: loadPage('Blog') }
]
```

Test the new page

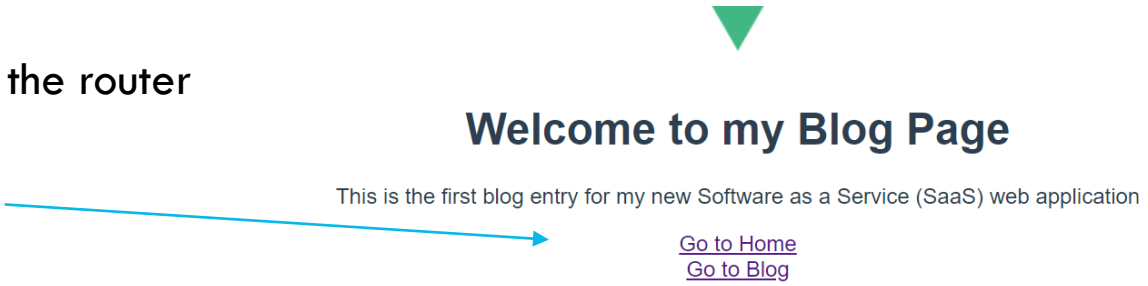
- Navigate to the URL http://localhost:5173/blog

Router Links <router-link>

- Vue supports a link component <router-link>
- Instead of normal <a> tags, Vue supports a custom link component which will resolve links automatically.

```
<template>
   <h1> This is where the links should be </h1>
   <router-link to="/">Go to Home</router-link>
   <br>
   <router-link to="/blog">Go to Blog</router-link>
</template>
```

Two example links using the router link component



Boot up a local server and test

- Now if you execute the command npm run serve the pages should load up at the various paths "/" and "/blog"
- You have successfully implemented a client side router and programmed in routes to specific components or pages
- If you generate a build and deploy it will also work.

Exercise 2 – Add a nav bar

- Add a new component called **Navigation.vue** which will contain navigation links for you application
- Place two links in navigation component -> Home, Blog
  - Use router links for the two pages.
- Add an additional page called About Us
  - Add a route to the page to the routes.js file with path /about-us
  - Add a link in the Navigation.vue component to the About Us page
- Deploy to Firebase

Routing summary

- This was a basic introduction to routing with the vue-router package, there is a lot more information available on how to configure much more advanced routing on the router information page
  - https://router.vuejs.org/guide/

VUEJS – INVOKING SERVERLESS FUNCTIONS

Overview

- Adding Bootstrap to our Vue App
- Creating a simple comment form on the blog page
- Making our first client call to post data to the server
  - Client-side API for function calls
    - Background on AJAX
  - SDK setup and configuration
  - Data binding in VueJS
  - Methods in VueJS
  - Same Origin Policy
  - CORS

Adding a design framework

- There are a number of projects with tailored Design Frameworks for VueJS
- One is called BootstrapVue
  - https://bootstrap-vue.org/

BootstrapVue

- Also a Material Design package called Vuetify
  - https://vuetifyjs.com/en/

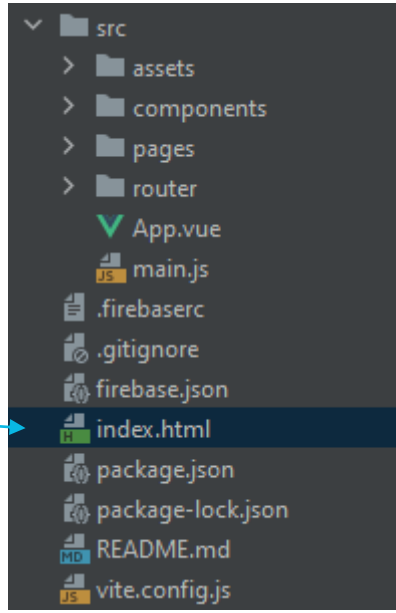
Adding Bootstrap

- We can manually add Bootstrap to our applications similarly to how we did last semester.
- Go to https://getbootstrap.com/docs/5.3/getting-started/introduction/

Where to place the URLs?

src/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
crossorigin="anonymous">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-
w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CX
vN" crossorigin="anonymous"></script>
      <title>My Application</title>
  </head>
  <body>
    <div id="app"></div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```



Running example – Blog comments

- In order to work through this weeks' content, I'm going to develop a simple comment form that we pop at the bottom of the blog page. The idea being that app users can post comments when there are new blog entries.

- This will require the saving of new comments into the database, the requesting and displaying of existing comments etc.

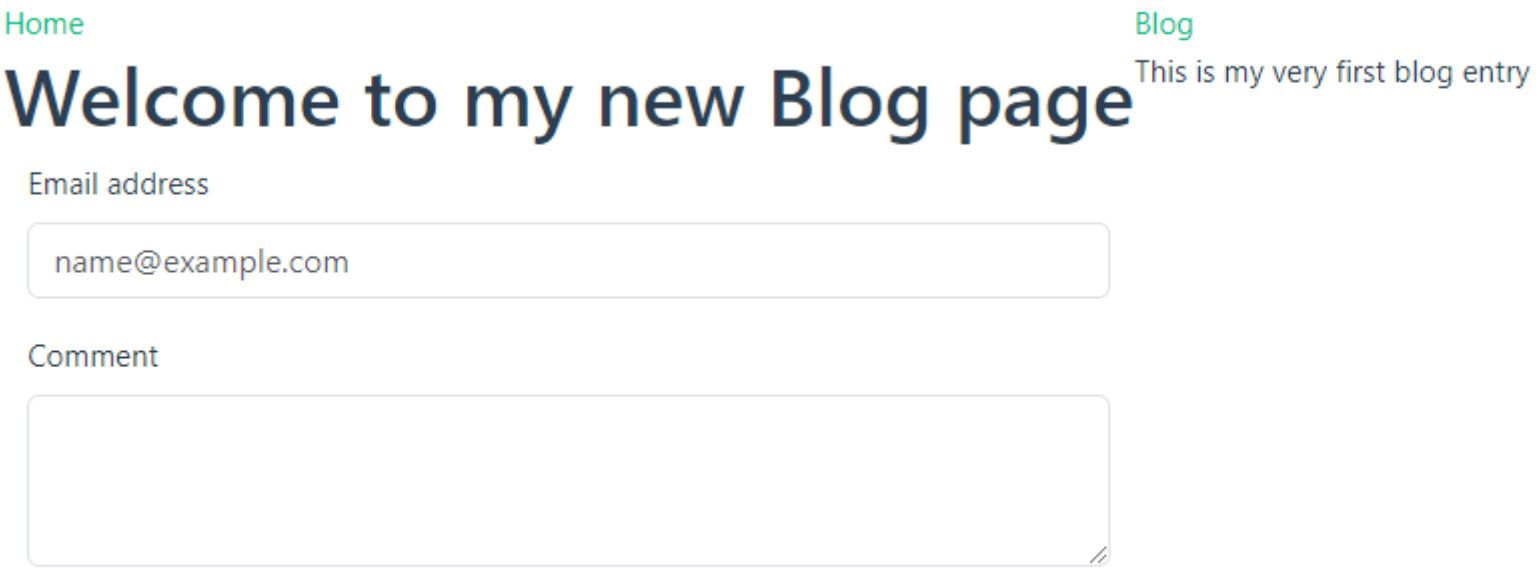
- This will teach us how to invoke our serverless functions via the client and connect to the database, to add new comments and read existing ones.

Add a comment form in Blog.vue

- First step is to add a comment form at the bottom of the existing blog page from last week. Sample code taken from https://getbootstrap.com/docs/5.1/forms/form-control/

```
<template>
<h1>Welcome to my blog page</h1>
 <p>This is my blog page for my SaaS app</p>
 <div class="container">
  <div class="mb-3">
   <label for="exampleFormControlInput1" class="form-label">Email address</label>
   <input type="email" class="form-control" id="exampleFormControlInput1"
placeholder="name@example.com">
  </div>
  <div class="mb-3">
   <label for="exampleFormControlTextarea1" class="form-label">Comment</label>
   <textarea class="form-control" id="exampleFormControlTextarea1" rows="3"></textarea>
  </div>
 </div>
</template>
```

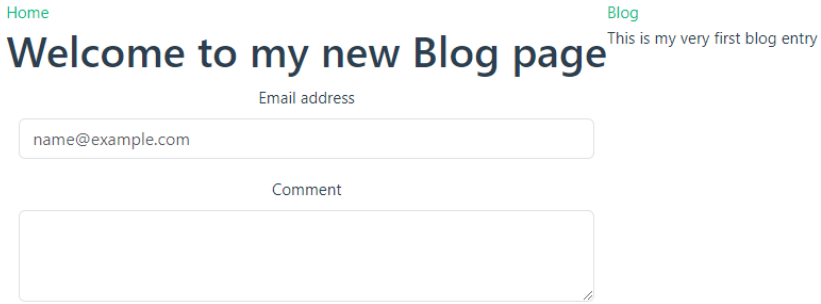
Resulting page



Adjusting the styles for the Blog.vue SFC

- With VueJS one can alter a specific component's SFC styling by specifying scoped styles in the <style> tags.
- The global text styling in App.vue specifies that the text should be centred. This can by overridden easily within the SFC.

```
<style scoped>
.container{
  text-align: center;
}
</style>
```



Add a button to the form

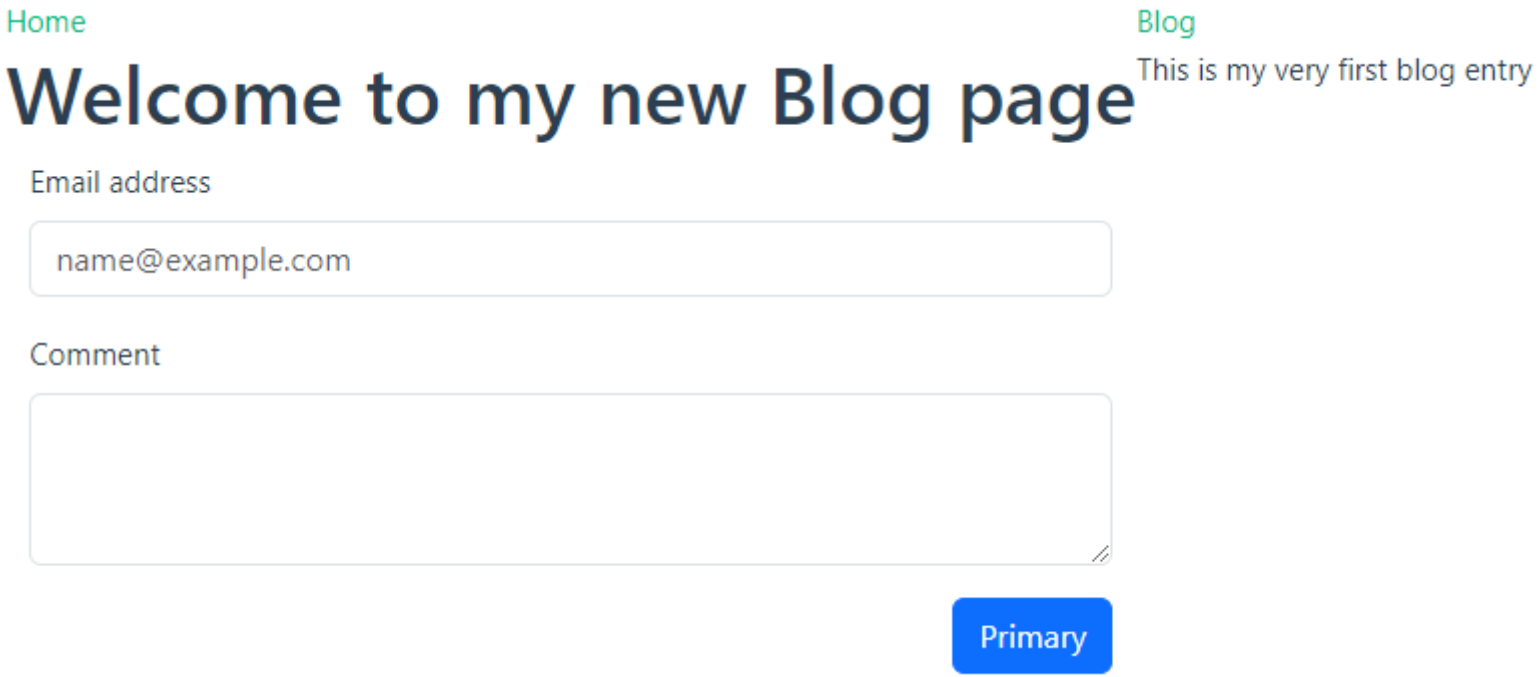

```
<template>
<h1>Welcome to my blog page</h1>
  <p>This is my blog page for my SaaS app</p>
  <div class="container">
    <div class="mb-3">
      <label for="exampleFormControlInput1" class="form-label">Email
address</label>
      <input type="email" class="form-control"
id="exampleFormControlInput1" placeholder="name@example.com">
    </div>
    <div class="mb-3">
      <label for="exampleFormControlTextarea1" class="form-label">Post
Comment</label>
      <textarea class="form-control" id="exampleFormControlTextarea1"
rows="3"></textarea>
    </div>
    <div class="mb-3 right">
      <button type="button" class="btn btn-primary">Primary</button>
    </div>
  </div>
</template>

<style scoped>
.right{
  text-align: right;
}
</style>
```

Simple Bootstrap Button

Aligning the button to the right

UI should now look like…



Exercise 1 – Add Bootstrap, create Form

- Add Bootstrap to your VueJS app (slide 5)
- Add a simple form as shown to add comments to our fictional blog page for our fictional SaaS
- Add the Bootstrap button “Post Comment” under the form

Firebase functions

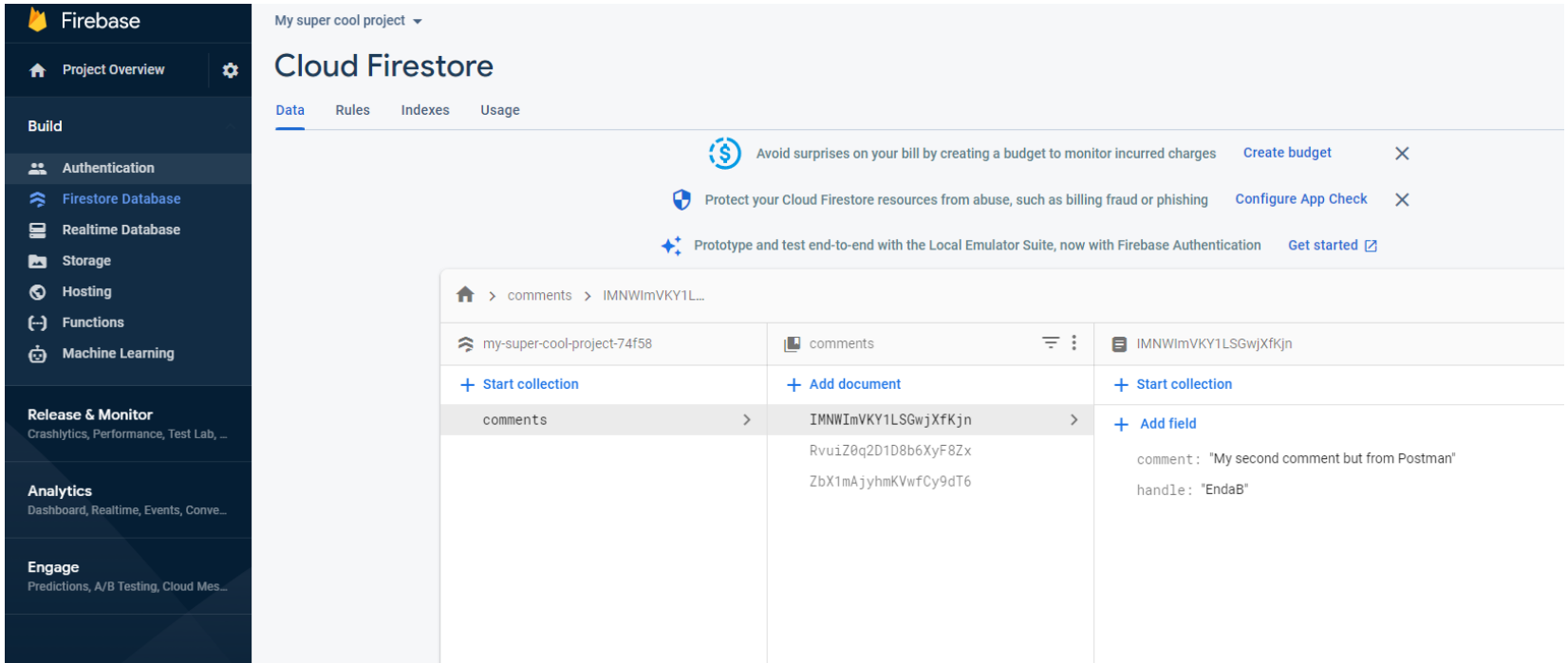
- At the end of semester 1 (Weeks 10 and 11) we looked at writing functions to write data to the database and read data back out.
- These were tested exclusively using the POSTMAN client
- In case you missed it or forgot it, I'm going to do a quick recap here.

Quick recap: On the process

- *Week 10 – Firebase functions, callbacks and creating our first functions*
- *Week 11 – Firestore Database*
- *Topics covered*
  - Creating a function to add a document
  - Using POSTMAN to test the function
  - Creating a fn to read documents
  - Testing document reads

Creating a Firestore DB

- Many of you are reusing the existing project from last semester so the DB is already created.
- If you have created a new project then login to Firebase, select your project from the console, click on Firestore and then Create Database



Adding a document


```
const functions = require("firebase-functions");
const admin = require('firebase-admin');
admin.initializeApp();

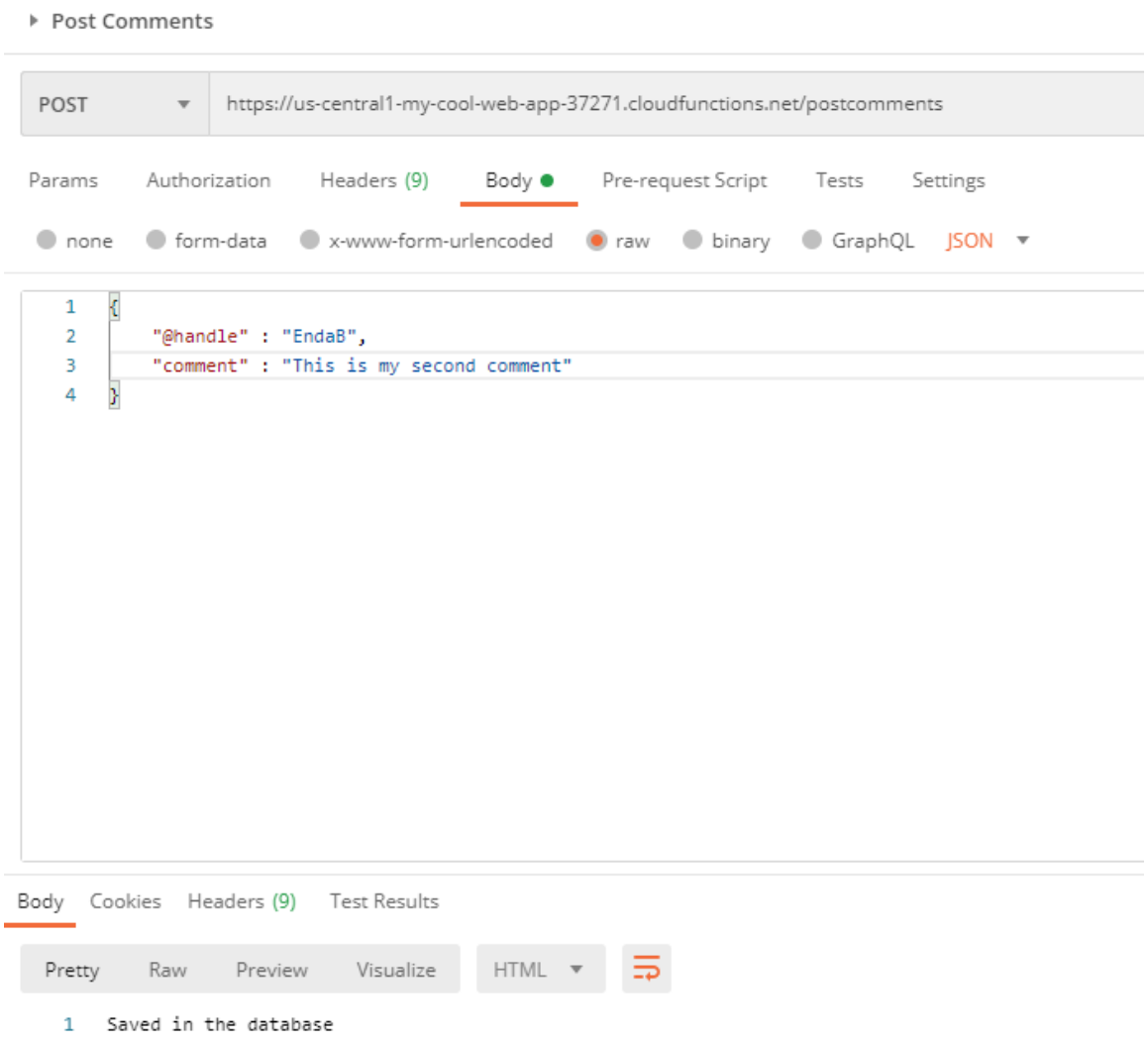
// Accept comment and return the same comment to the user
exports.postcomment = functions.https.onRequest((request,
response) => {

    const currentTime = admin.firestore.Timestamp.now();
    request.body.timestamp = currentTime;

    return
admin.firestore().collection('comments').add(request.body).then((
)=>{
        response.send("Saved in the database");
    });
});
```

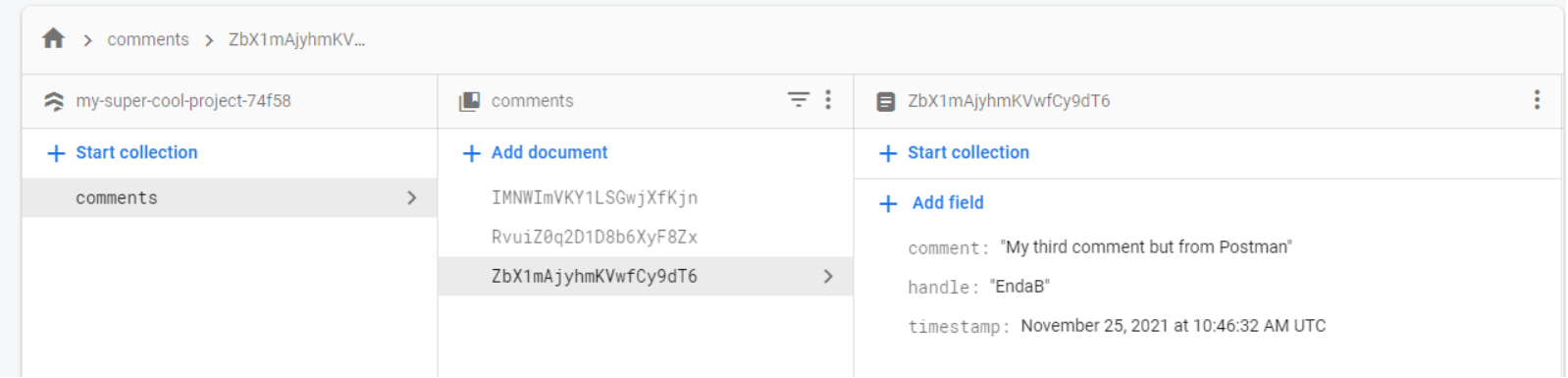
functions/index.js

Using POSTMAN POST to the fn



Check the database to see if it saved

- If you check on Firebase you should now see your comment



Exercise 2 – Functions

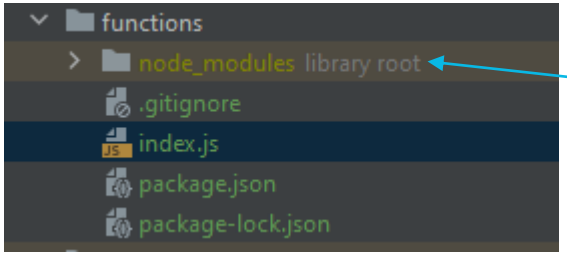
- Add a Firestore DB (if not already done)
- Deploy the function from last semester, “postcomment”
- The code for these (if you don’t already have it) is up on Blackboard under Week 15.
- Test that they work via POSTMAN calls, make sure that the comments are saving correctly in the database and the returning comments.

Server-side API for Firebase

- When creating functions you use the server-side Firebase library

```
const functions = require("firebase-functions");
const admin = require('firebase-admin');
admin.initializeApp();

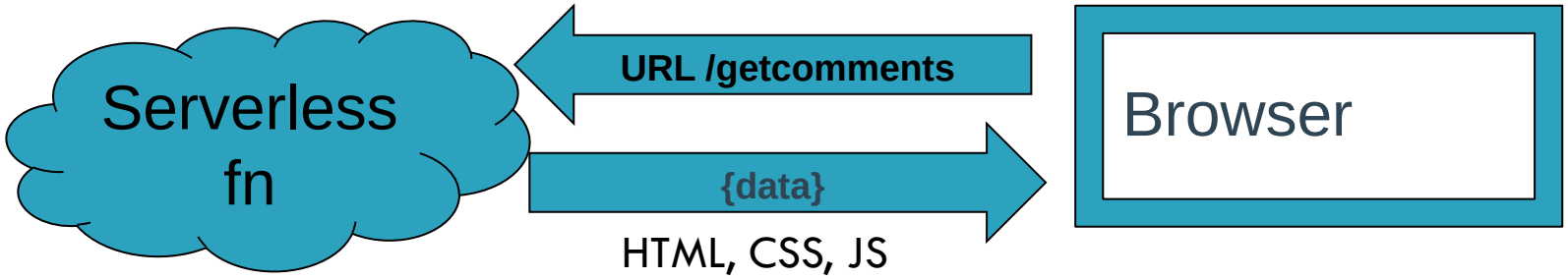
// // Create and Deploy Your First Cloud Functions
```



Separate node_modules located under functions, contain the library code

Client-side API

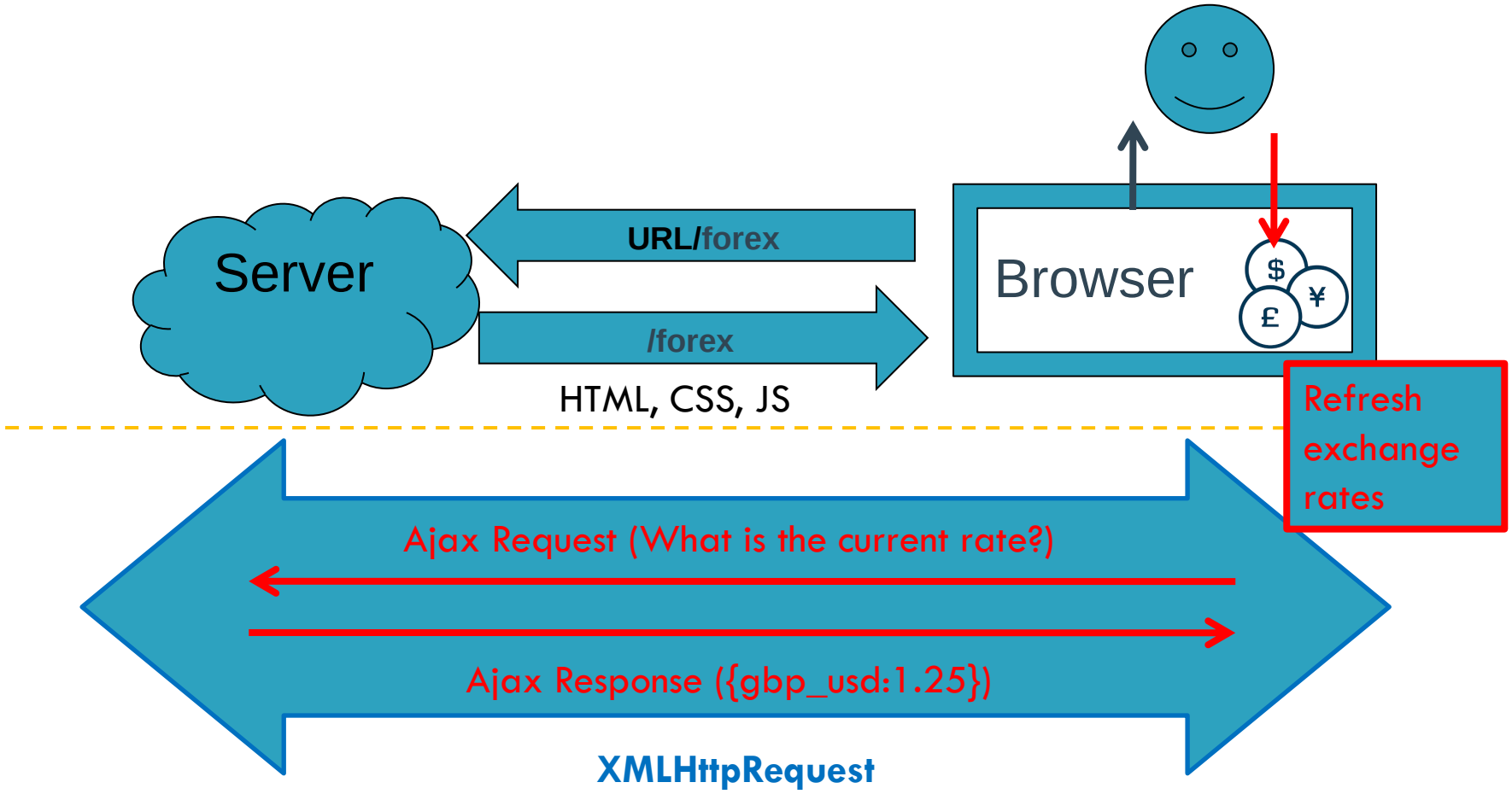
- To invoke our serverless functions “postcomment” and “getcomments” from the VueJS client application code we need to utilize the Firebase client-side APIs.
- This code enables us to call (invoke) the deployed serverless functions



What is AJAX?

- A key driver of Web 2.0
- Normal HTTP interactions instruct the browser to request an entire webpage at once
  - http://www.example.com/
- AJAX requests instruct the browser to request small pieces of content
  - Typically, smaller parts of the webpage
  - But can be any arbitrary data, not necessary marked up as HTML
  - Asynchronous update => no tangible 'refresh' process for user

AJAX Requests



What is AJAX?

- **A**synchronous
  - Request is made without interrupting the page
  - Often user initiated by events, sometimes timed
- **J**avaScript
  - Usually kicks in when the Response is ready, i.e. the server returns with the data
- **X**ML
  - Language agnostic mechanism for data transport and storage (Data Format)

However, it also supports JSON

- XML in the name can be somewhat misleading, given that you can send data as plain text or JSON etc.
- It's simply a combination of
  - XMLHttpRequest Object (browser built in)
  - JavaScript to manipulate the DOM
- We will be using the JSON data format for our data
- AJAJ!

XMLHttpRequest

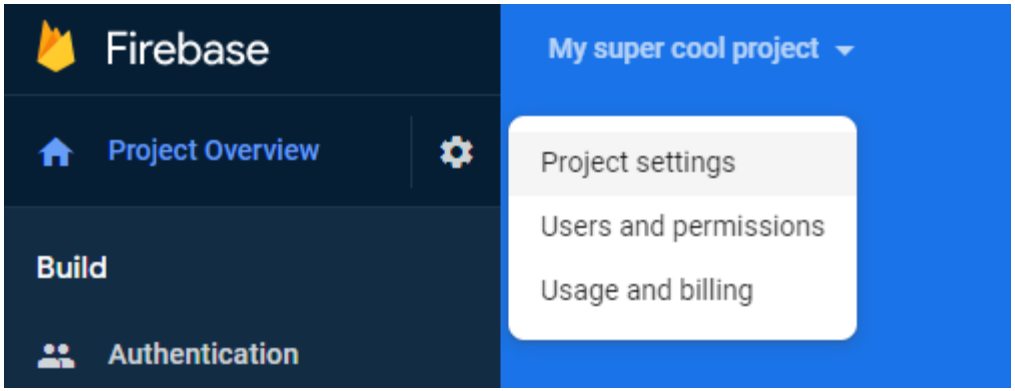
- All modern browsers have a built in XMLHttpRequest object
- Allows us to make a request to the server without going through normal HTTP request process
  - i.e. by popping the URL in the address bar of the browser
- Thus we can use this make calls to our Cloud Functions, however Firebase has a separate library which we can use instead which keeps our apps within a single ecosystem

Install the Firebase client SDK

- Open a terminal or cmd window
- Navigate to the root project directory and execute the command npm install firebase
- This will install the firebase client SDK, which will enable us to start invoking our functions from our client code (VueJS code)

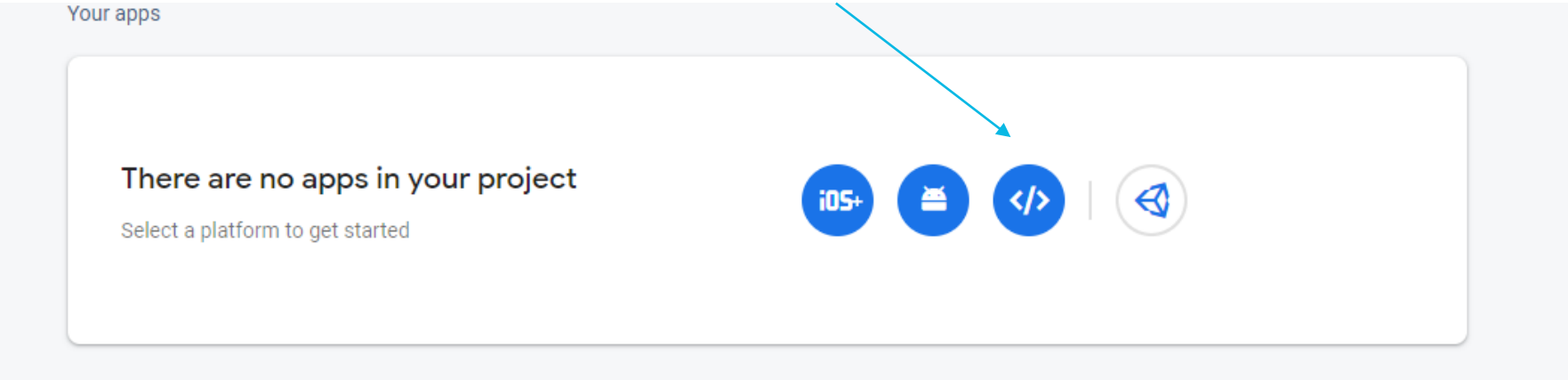
Initialise the client code

- Before we can make any calls to our functions though we must initialize the client code.
- Login to the Firebase dashboard and click the settings Icon and then project settings under your application.

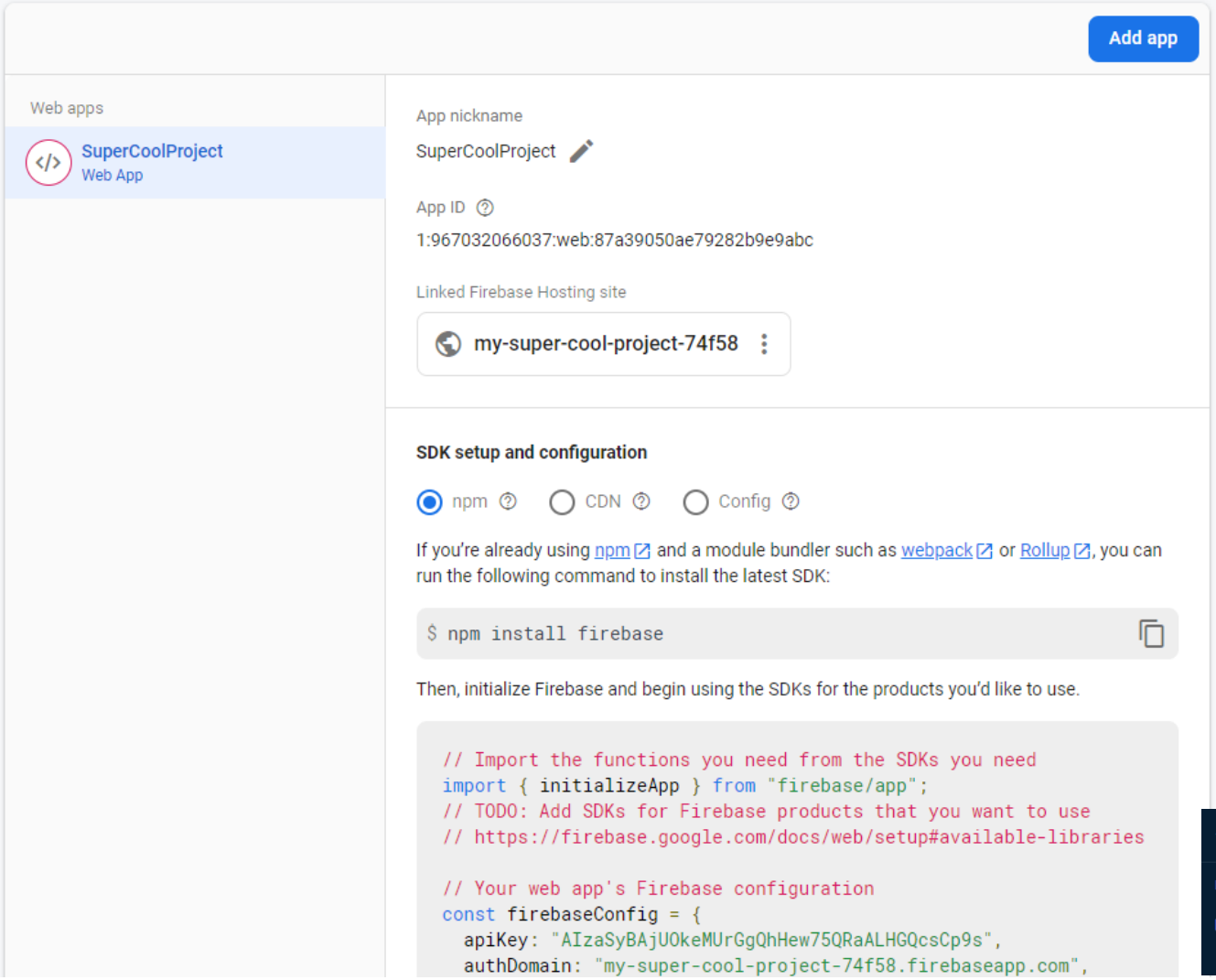


SDK Setup and Configuration

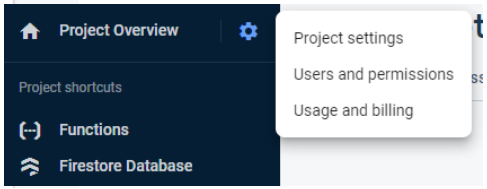
- If you've already added web hosting the configuration should be available, otherwise just click on the web icon and add it
- Follow the sequence of steps (note: we already installed firebase client library on slide 27)



SDK Setup and Configuration

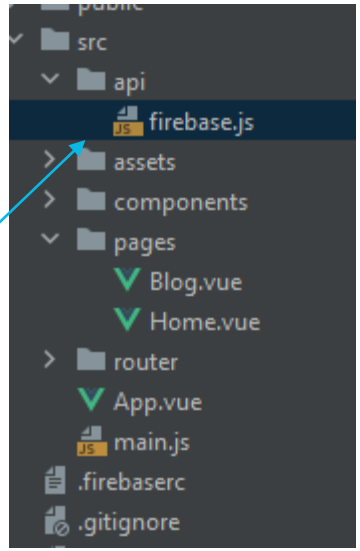


Should now see the configuration info for the client SDK, listed at the bottom of the project settings page



Add a directory “api” under src

- Under the src folder create a new directory called api



- Add a new file called firebase.js
- In this file we’ll place the configuration info listed on the firebase dashboard for your app and export it as a module which can be imported to other files

Firebase.js

- Open the file firebase.js and copy the config code from Firebase into the file

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "[redacted]",
  authDomain: "my-super-cool-project-74f58.firebaseapp.com",
  projectId: "my-super-cool-project-74f58",
  storageBucket: "my-super-cool-project-74f58.appspot.com",
  messagingSenderId: "967032066037",
  appId: "1:967032066037:web:87a39050ae79282b9e9abc"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Export the app

- Add an additional line to the firebase.js file at the end to export “app” as the default export for the module

```
  messagingSenderId: "967032066037",
  appId: "1:967032066037:web:87a39050ae79282b9e9abc"
};


// Initialize Firebase
const app = initializeApp(firebaseConfig);

export default app;
```

Exercise 3

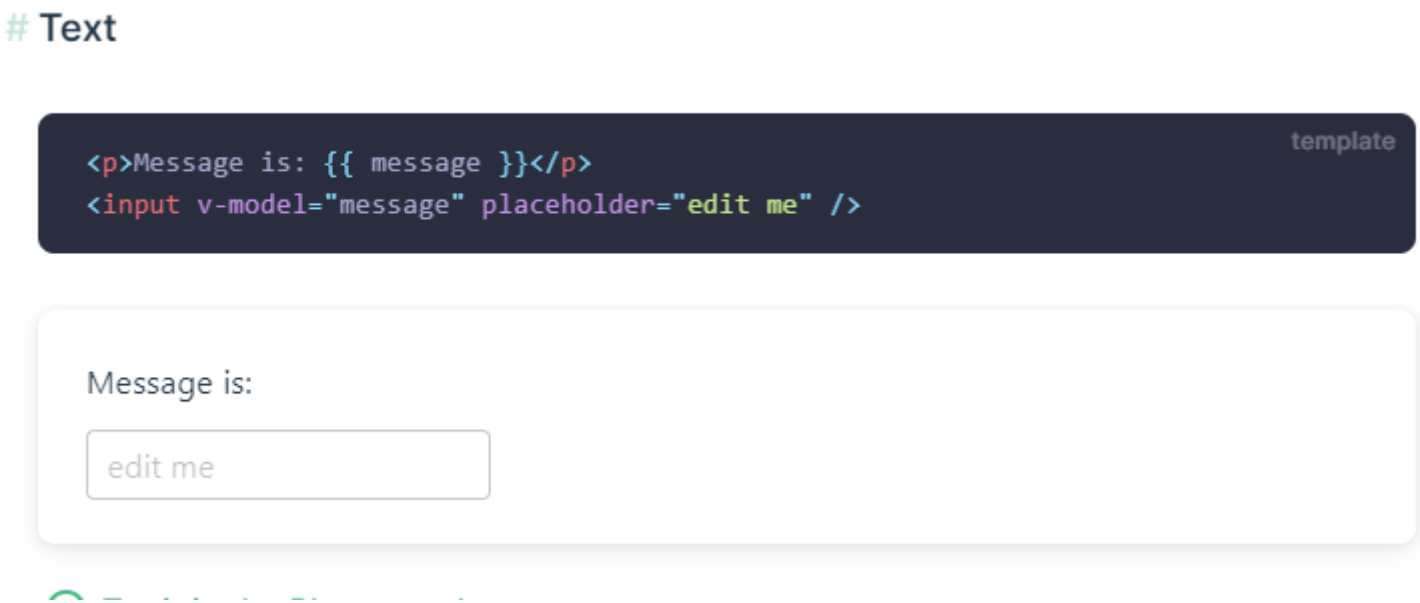
- Add the Firebase client SDK to our apps (slides 27-33)

Data binding

- Two-way data binding in VueJS.
- Vue supports a concept known as two-way data binding.
- Put simply when the user types into the input field, the bound variable gets updated to match the value in the input. Equally when you update the bound variable, the input element's content updates to match value.

Data binding in VueJS

- A directive known as v-model can be applied to input tags, i.e. inputs, textarea's and select elements to create a two-way data binding



https://vuejs.org/guide/essentials/forms.html#text

Data binding in VueJS

- Add two properties “handle” and “comment” as v-model attributes on the two inputs for our form

```
<div class="mb-3">
  <label for="exampleFormControlInput1" class="form-label">Email address</label>
  <input type="email" v-model= "handle" class="form-control" id="exampleFormControlInput1"
placeholder="name@example.com">
</div>
<div class="mb-3">
  <label for="exampleFormControlTextarea1" class="form-label">Comment</label>
  <textarea class="form-control" v-model="comment" id="exampleFormControlTextarea1"
rows="3"></textarea>
</div>
<div class="mb-3 right">
  <button type="button" class="btn btn-primary">Post Comment</button>
</div>
```

Data option for a component

- In slide 36 the data entered by the user into the text box was immediately displayed on the client.
- How do we access the data in our JS code, i.e. between the script tags on a SFC.

```
<script>
export default {
 data() {
  return {
   handle: '',
   comment: ''
  }
 },
}
</script>
```

These two properties “handle” and comment are bound to the inputs via the v-model directive (previous slide)

As the user types into the text boxes, these will be populated with the value in real-time and vice versa (if we populate via JS). This is two-way binding.

Initialised to contain no value, empty string, i.e. two single quotes ‘’

https://vuejs.org/guide/essentials/reactivity-fundamentals.html

Methods in VueJS

- Vue allows for the creation of methods. Put simply these are functions we can attach to handle events, i.e. button clicks etc.

- If we define a method which will run when the user clicks the “post comment” button we can then take the data from the input form and post it to the server!

https://v3.vuejs.org/guide/data-methods.html#methods

Method stub to post comments


```
<div class="mb-3 right">
    <button type="button" @click="postComment" class="btn
btn-primary">Post Comment</button>
  </div>
</div>
</template>

<script>
export default {
data() {
   return {
     handle: '',
     comment: ''
   }
 },
 methods : {
   postComment() {
     console.log(this.handle);
     console.log(this.comment);
   }
   }
}
</script>
```

@click directive is equiv to onclick and specify the method postComment that is invoked onclick.

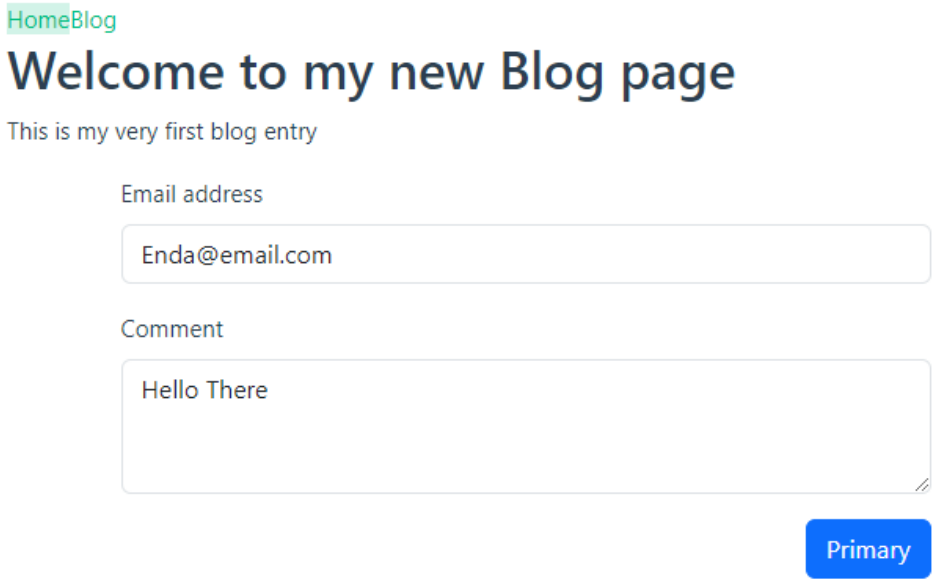
Define a function called postComment() {…}, placed under methods which logs to the console the values in *handle* and *comment*

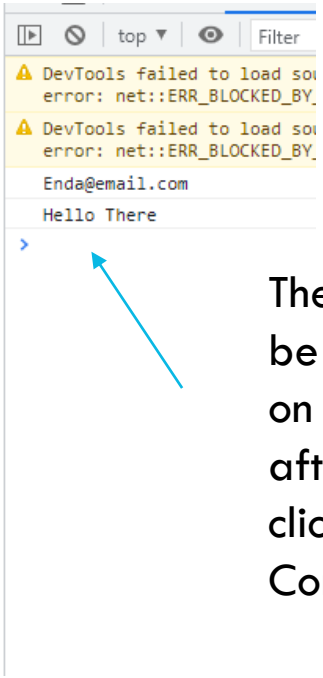
Once the user types into the text boxes the values for *handle* and *comment* get populated

The "this" keyword must be used to reference them

Open up localhost:5137

- Open the developer tools tag and check the console.
- Type into the boxes and click Post Comment





The data should be visible here on the console, after you have clicked Post Comment

Making calls to the server

- At this point the Blog page has a simple form where the user can enter their “handle” and leave a comment.
- The data entered into the text boxes is stored in the two member variables.
- When the user clicks on the button “Post Comment” the method “postComment()” is called.
- The method currently just writes the values of the variables to the console but we’d like to instead send this data to the server!

Adding the Client SDK to the SFC


```
<script>
import app from '../api/firebase';
import { getFunctions, httpsCallable } from "firebase/functions";

export default {
 name: "Blog",
 data() {
  return {
   handle: '',
   comment: ''
  }
 },
 methods : {
  postComment () {
   console.log(this.handle);
   console.log(this.comment);
   const functions = getFunctions(app);
   const postComment = httpsCallable(functions, 'postcomment');
   postComment({"handle": this.handle, "comment":
this.comment})
      .then((result) => {
       // Read result of the Cloud Function.
       /** @type {any} */
       console.log(result);
      });
  }
  }
}
</script>
```

Import the exported app from api/firebase with the app config.

Import the objects getFunctions and httpsCallable from firebase/functions

Pass in the app config as a param to getFunctions

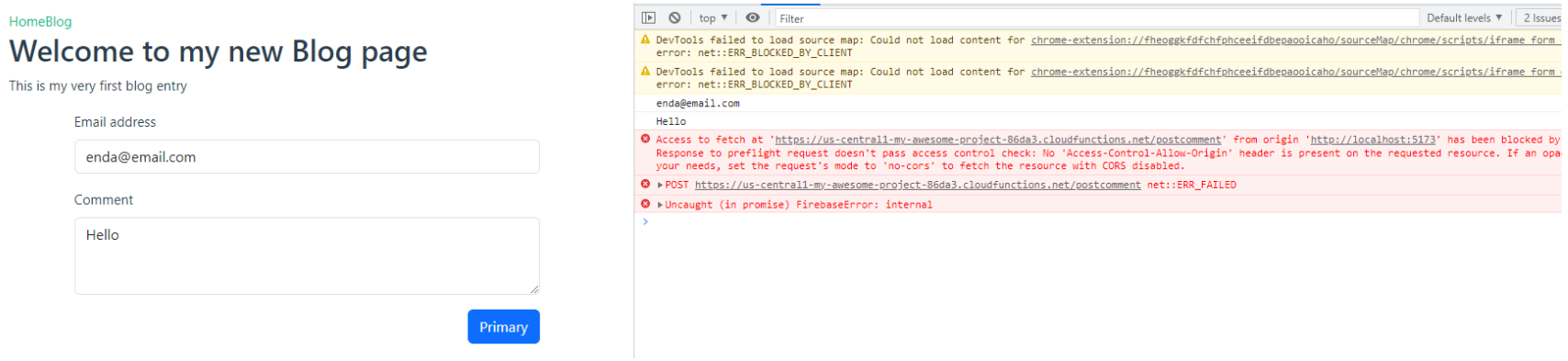
Using httpsCallable, pass in functions and the name of the function you wish to call

Pass the object as an argument to postComment

Open localhost:5137

- Open up the developer tools and check the console
- Type into the boxes and click Post Comment

CORS Policy - Error



What is CORS?

- Firstly lets talk about SOP or Same Origin Policy
- For security reasons browsers prevent scripts from one origin accessing resources from another origin.
  - This is known as the same origin policy.
- Two URLs have the same origin if the protocol, port (if specified), and host are the same for both.

Examples of SOP

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

CORS

- Cross Origin Resource Sharing
  - CORS allows a script executing in one domain access a resource in a different domain

- Browsers make a “preflight” request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request.

- In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

Hosting and Functions

- Our Firebase hosting and functions violate the SOP
- Lets examine the URLs to find out why
- https://my-super-cool-project-74f58.web.app:80/
- https://us-central1-my-super-cool-project-74f58.cloudfunctions.net/postcomment
- The domain/host names are different

How do we circumvent SOP?

- Luckily we can enable our functions to violate the SOP using by using CORS
- Note that there are security implications of doing this.
- There is a NodeJS library called cors that we can use to enable the feature
- Navigate to the functions directory within your app on the command line and execute the command
  - npm install cors

Adding CORS library to server fn


```
const functions = require("firebase-functions");
const admin = require('firebase-admin');
const cors = require('cors')({origin: true});

admin.initializeApp();

// Accept comment and return the same comment to the user
exports.postcomment = functions.https.onRequest((request, response) => {
cors(request, response, () => {

    const currentTime = admin.firestore.Timestamp.now();
    request.body.timestamp = currentTime;

    return
admin.firestore().collection('comments').add(request.body).then(()=>{
        response.send({"data":"Saved in the database"});
    });
  });
});
```

functions/index.js

Remember to redeploy the functions once you have made the changes!

Communication established

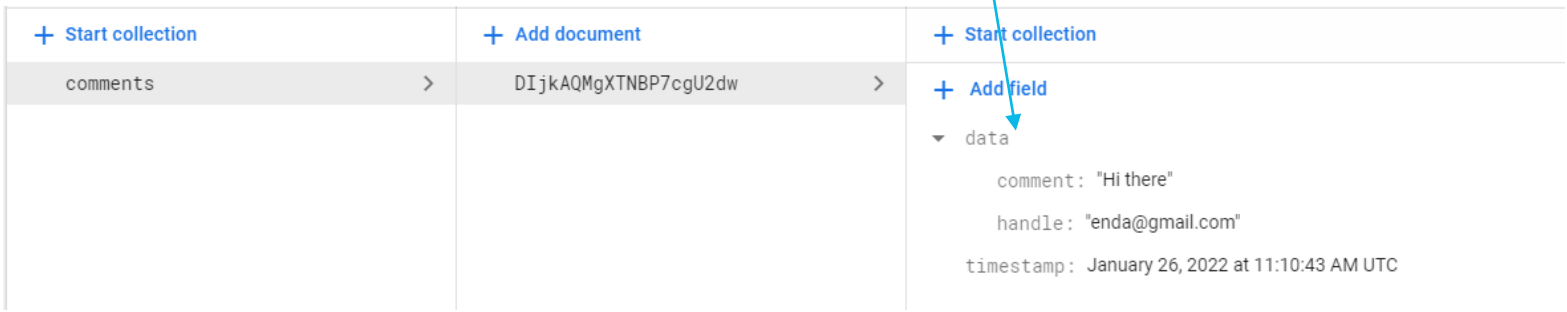
- Now that CORS is enabled on the server we can communicate with it from our client side JS

- Note that origin true will allow requests from any origin. This is generally a bad idea…

- ```
  const cors = require('cors')({origin: true});
  ```
  ```

Check the database

- You will see an entry now, but the posted data is nested within a new parent “data” property. This is automatically added when calling via the client libs




```
const functions = getFunctions(app);
const postComment = httpsCallable(functions, name: 'postcomment');
postComment( data: {"handle": this.handle, "comment": this.comment})
    .then((result) => {
```

Modify the function to remove “data” prop

- If you wish to remove it, you can modify the function as follows

```
exports.postcomment = functions.https.onRequest((request,
response) => {
   cors(request, response, () => {

      const currentTime = admin.firestore.Timestamp.now();
      request.body.timestamp = currentTime;
      return admin.firestore().collection('comments').add({
handle: request.body.data.handle,
         comment:request.body.data.comment, timestamp:
request.body.timestamp}).then(() => {
            response.send({"data": "Saved in Database"});
      });
   });
});
```

functions/index.js

Exercise 4

- Following the slides implement the code to post a comment from your blog page to your deployed server function, which in turn saves the data to the database.

CONTINUING WITH CRUD

Dr. Enda Barrett

Lecture overview

- Retrieving comments from our Firestore Database via our serverless function getComments
- Serverless delete function and client code to delete comments
- Serverless update function and client code to update comments

Get comments and display them

- The getComments serverless function should already be in place from last semester. If not you can find the code on Blackboard under Week 15

```
exports.getcomments = functions.https.onRequest((request,
response) => {
   cors(request, response, () => {
      // 1. Connect to our Firestore database
      let myData = []
      admin.firestore().collection('comments').orderBy("timestamp",
"desc").get().then((snapshot) => {

         if (snapshot.empty) {
            console.log('No matching documents.');
            response.send('No data in database');
            return;
         }

         snapshot.forEach(doc => {
            myData.push(doc.data());
         });

         // 2. Send data back to client
         response.send({data : myData});
      });
   });
});
```

Make sure cors is enabled

functions/index.js

Data property

Create client code to display it

- Goal is to add some code to Blog.vue to display all the comments.

```
getComments(){
   const functions = getFunctions(app);
   const getComments = httpsCallable(functions,
'getcomments');
   getComments().then((result) => {
     // Read result of the Cloud Function.
     // /** @type {any} */
     console.log(result);
   });
  }
}
```

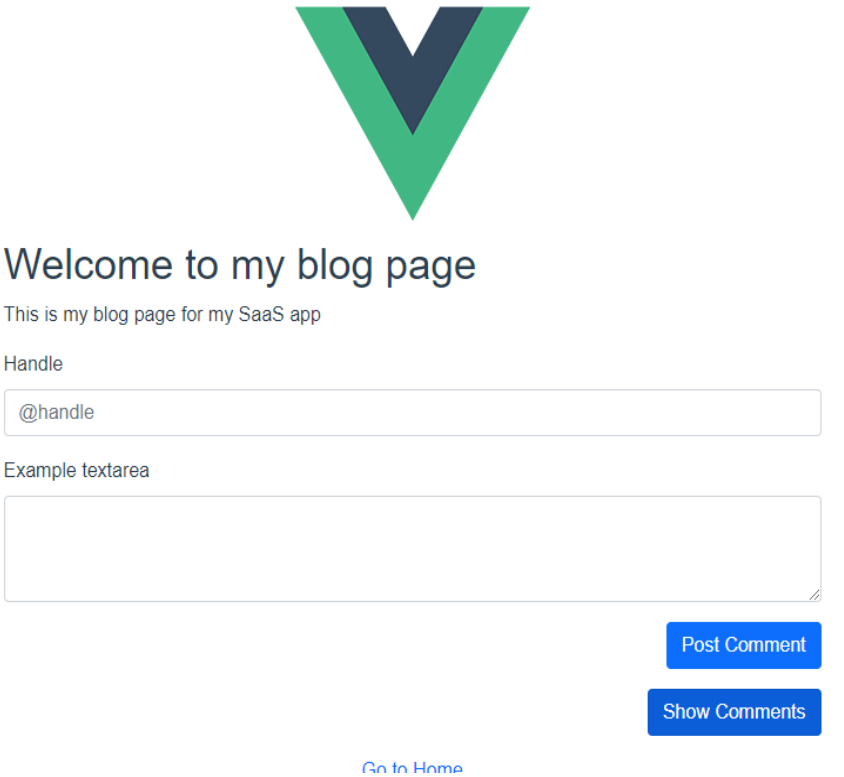
Add a method under postComment called getComments

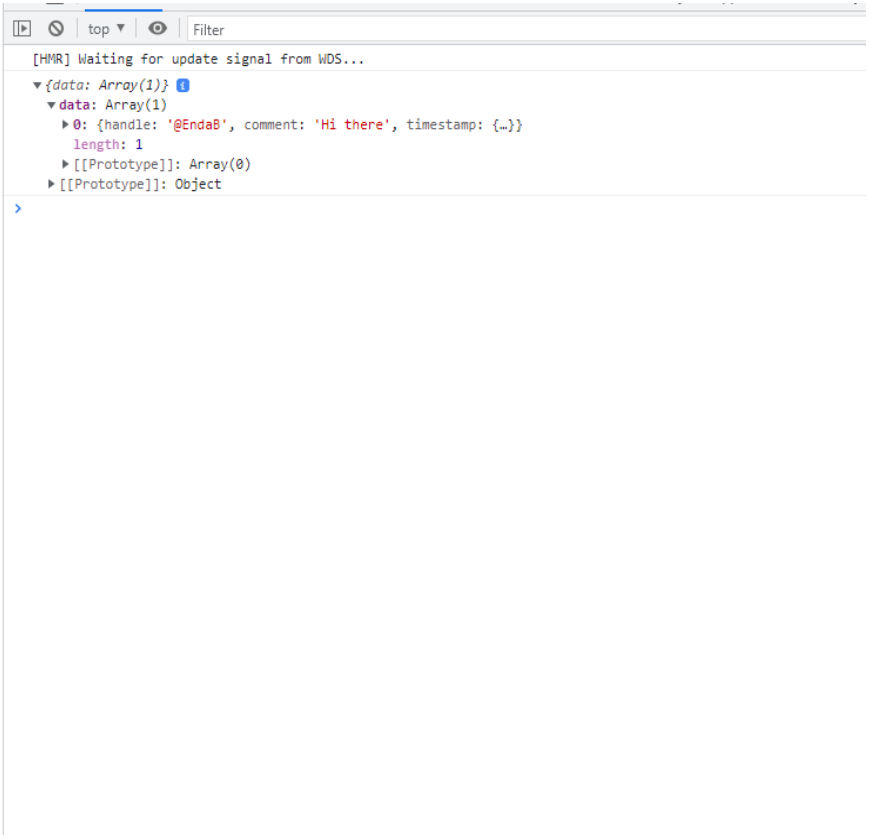
Blog.vue

Add a show comments button, in the template

```
<div class="mb-3 right">
  <button type="button" @click="getComments" class="btn btn-primary">Show Comments</button>
</div>
```

Click on Show Comments





List rendering in VueJS

- VueJS supports a v-for directive which allows us to loop over an array within our Vue code and display items.

```js
data() {
  return {
    items: [{ message: 'Foo' }, { message: 'Bar' }]
  }
}
```

```template
<li v-for="item in items">
  {{ item.message }}
</li>
```

https://vuejs.org/guide/essentials/list.html#list-rendering

List rendering of comments

- Add the following below the Show Comments button

```
<div class="mb-3">
 <ul id="array-rendering">
   <li v-for="comment in
comments">
     {{ comment.comment }}
   </li>
 </ul>
</div>
```

- Add an empty “comments” array to the data object

```
data(){
 return {
   handle:'',
   comment:'',
   comments:[]
 }
},
```

Show comments

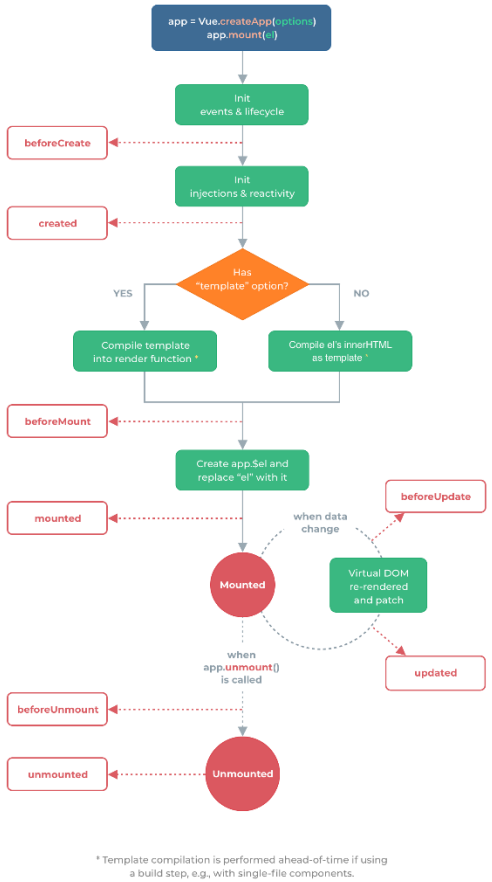
- When show comments is clicked a request is made for the comment data, this is assigned to the comments array. VueJS will render whatever data is placed in the array, as there is a 2-way binding established on page load

```
getComments(){
const functions = getFunctions(app);
const getComments = httpsCallable(functions,
'getcomments');
 getComments().then((result) => {
   // Read result of the Cloud Function.
   // /** @type {any} */
   console.log(result);
   this.comments = result.data;
 });
}
```

Mapping the response data to the comments array

Show comments on page load

- Loading on button click is fine, but sometimes you may wish to show comments on page load.
- When the SFC loads for the first time, there is a specific lifecycle that occurs where a number of functions are run in a specific order.



https://stopbyte.com/t/what-are-lifecycle-hooks-in-vuejs-and-how-do-they-work/1009

Load comments on page load

- In the current blog page, the comments don't load when the page loads. Instead, one must click the "show comments" button.
- However, if we utilize the created function, we can have the comments display as soon as the page loads.

```
export default {
  name: "Blog",
  data(){
    return {
      handle:'',
      comment:'',
      comments:[],
      tempValue:"",
      editing:false
    }
  },
  created(){
    this.getComments()
  },
  methods: {...}
}
</script>
```

Exercise 1: Get comments

- Following the steps in the previous slides 3-10, retrieve the comments from the database by invoking the getcomments serverless function and display them on the client.

- Using the “created” function, invoke getcomments on page load and display the comments when the page loads for the first time.

- **Advanced:** Using the window.setInterval method request comments every 30 seconds

Loading…

- When you click on the show comment or post comment there will be a delay during the function call process.
- It would be useful to indicate to the user once they click on the button that a request is underway, and they should wait until it comes back before pressing the button again.
- Functions can be very slow when they are cold! 5 second response time.

Vue-loading-overlay

- There are many of these but a nice one that I found is called vue-loading-overlay
- Execute npm install vue-loading-overlay in the root of your app folder

```
import { createApp } from 'vue'
import App from './App.vue'
import { createRouter, createWebHistory } from 'vue-router';
import routes from './router/routes';
import Loading from 'vue-loading-overlay';


let router = createRouter({
    history: createWebHistory(),
    routes: routes
});

const app = createApp(App);
app.use(router);
app.use(Loading);
app.mount('#app');
```

Import the component

Main.js

Instruct Vue to use it

Method getComments – Blog.vue


```
getComments(){
  const functions = getFunctions(app);
  if(window.location.hostname === "localhost") // Check if working
locally
    connectFunctionsEmulator(functions, "localhost", 5001);
  const getComments = httpsCallable(functions, 'getcomments');
  let loader = this.$loading.show({
    // Optional parameters
    loader: 'dots',
    container: this.$refs.container,
    canCancel: false
  });
  getComments().then((result) => {
    // Read result of the Cloud Function.
    // /** @type {any} */
    // once the response has returned hide the loader
    loader.hide();
    console.log(result);
    if(result.data === 'No data in database')
      this.comments = [{comment : "No comments posted yet"}]
    else
      this.comments = result.data;
  });
}
```

Loader – Specified dots, the container and the set cancel to false

Hide the loader, once the function returns

Available config props

- https://www.npmjs.com/package/vue-loading-overlay

- Info on installation and configuration

Available props

The component accepts these props:

Attribute	Type	Default	Description
active	Boolean	`false`	Show loading by default when `true`, use it as `v-model:active`
can-cancel	Boolean	`false`	Allow user to cancel by pressing ESC or clicking outside
on-cancel	Function	`()=>{}`	Do something upon cancel, works in conjunction with `can-cancel`
is-full-page	Boolean	`true`	When `false`; limit loader to its container^
transition	String	`fade`	Transition name
color	String	`#000`	Customize the color of loading icon
height	Number	*	Customize the height of loading icon
width	Number	*	Customize the width of loading icon
loader	String	`spinner`	Name of icon shape you want use as loader, `spinner` or `dots` or `bars`
background-color	String	`#fff`	Customize the overlay background color
opacity	Number	`0.5`	Customize the overlay background opacity
z-index	Number	`9999`	Customize the overlay z-index

Completing CRUD

- Currently we have serverless functions and client code which supports creating comments and reading (displaying) comments
- There is a concept called CRUD (Create, Read, Update and Delete) when it comes to functionality for DB driven web applications
- The final two, “update and delete” can be implemented similarly to create and read
- In order to do so we must create two new functions and also the client code – let’s start with delete

1) Add new firebase function - delete

http://localhost…?id

```
exports.deletecomment = functions.https.onRequest((request, response) => {
        cors(request, response, () => {
        // deletes a comment using the id of the document
        admin.firestore().collection("comments").doc(request.query.id).delete().then(function()
        {
                response.send({"data", "Document successfully deleted"});
        })
    });
});
```

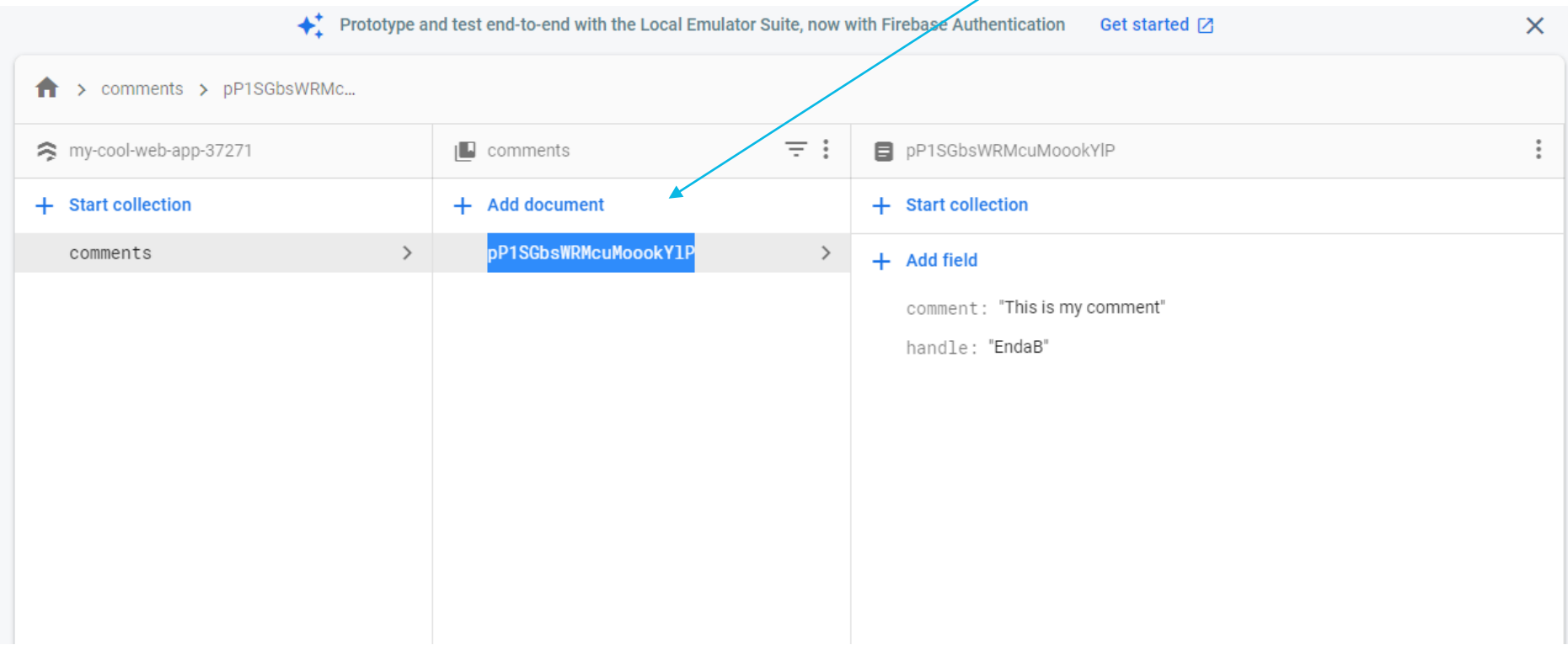
Add this code below your existing postcomment and getcomments functions

functions>index.js

Test your API works

1. Deploy your newly created function deletecomment to the cloud or alternatively run it locally on the emulator

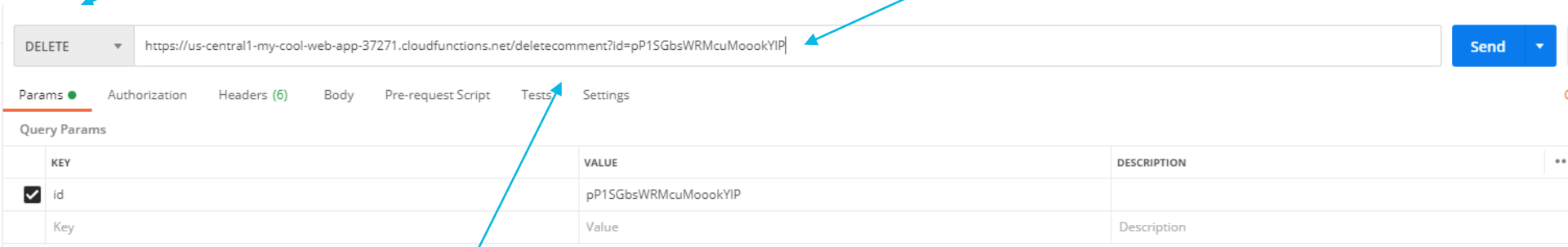
2. Open the database and get the ID of a document in the db



Postman DELETE request

Change request type to DELETE

Paste the id of the document you wish to delete



URL – http://..../deletecomment?id=sposifhak

Double check on the firebase console to be sure that it is gone from the database

Adding a delete button on the client


```
<div class="mb-3">
  <ul id="array-rendering">
    <li v-for="comment in comments">
      {{ comment.comment }}
      <button type="button" class="btn btn-primary">Delete
Comment</button>
    </li>
  </ul>
</div>
```

- This is my second comment Delete Comment
- This is my first comment Delete Comment

src>Blog.vue

Getting the document ID

- In order to pass the comment id to the serverless function via a delete call, the client code needs to have access to that id
- Currently getComments returns an array of comments, containing the handle, timestamp and comment itself.
- If we add the id of the document to this also, then we can pass it back when trying to delete comments

Modify getComments


```
exports.getcomments = functions.https.onRequest((request,
response) => {
   cors(request, response, () => {
      // 1. Connect to our Firestore database
      let myData = []
      admin.firestore().collection('comments').orderBy("timestamp",
"desc").get().then((snapshot) => {

         if (snapshot.empty) {
            console.log('No matching documents.');
            response.send({data: 'No data in database'});
            return;
         }

         snapshot.forEach(doc => {
            myData.push(Object.assign(doc.data(), {id:doc.id}));
         });

         // 2. Send data back to client
         response.send({data : myData});
      });
   });
});
```

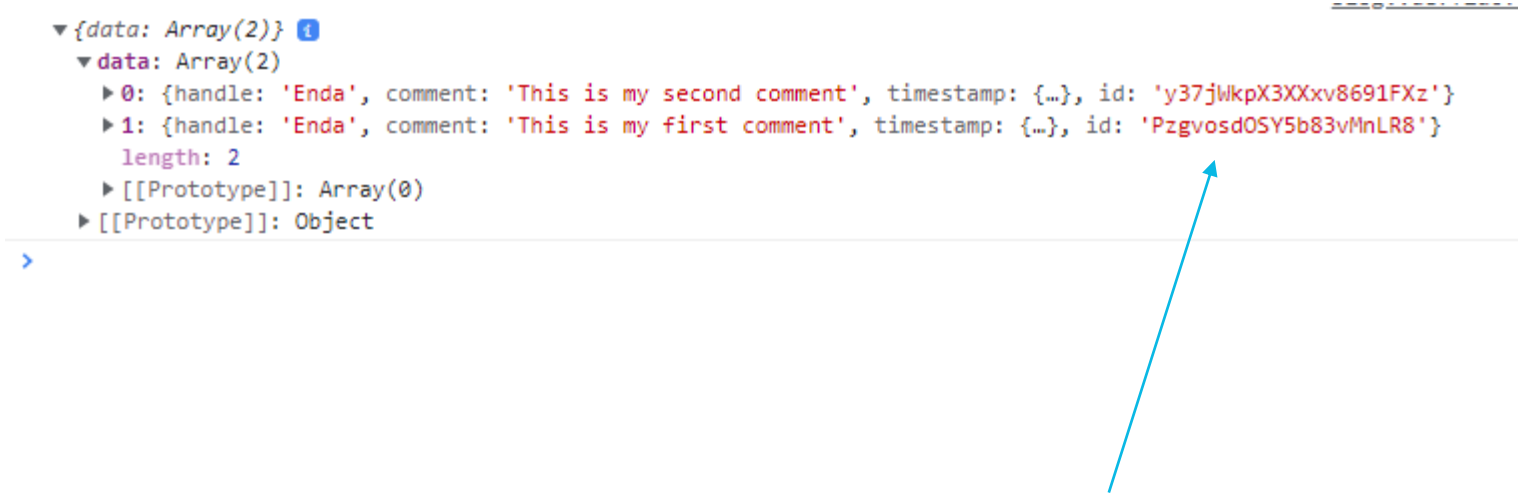
ECMAScript 2015 (ES6) supports an Object.assign method when you wish to merge two objects

doc.id contains the id of the document

functions>index.js

The client view

- On the client, if you click on Get Comments, you will see the following in the console…



Each object now contains the document id

Pass the id as a param

Modify the delete button to call a method deleteComment when the user clicks it

```
<div class="mb-3">
  <ul id="array-rendering">
    <li v-for="comment in comments">
      {{ comment.comment }}
      <button type="button" @click="deleteComment(comment.id)"
class="btn btn-primary">Delete Comment</button>
    </li>
  </ul>
</div>
```

Pass in the newly available id of each comment

src>Blog.vue

Create a deleteComment method

Pass in the doc id as a query

```
deleteComment(id){
  const functions = getFunctions(app);
const deleteComment = httpsCallable(functions, 'deletecomment?id='+id);
  deleteComment().then((result) => {
    if(result.data == "Deleted document from database")
      this.getComments() // To refresh the client
    })
}
```

Refreshes the UI by calling getComments() again

src>Blog.vue

Updating comments

Need the document ID passed in via the query

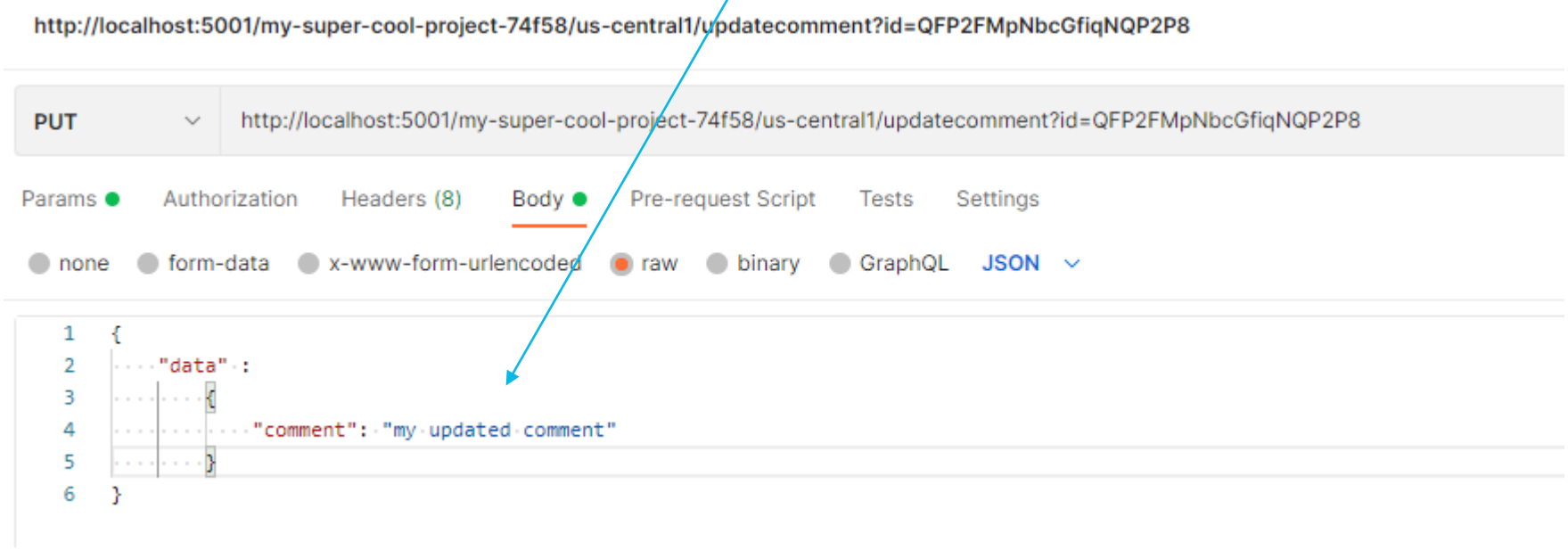
Need to supply the data you wish to update in the body of the request

```
exports.updatecomment = functions.https.onRequest((request, response) => {
   cors(request, response, () => {
      return
admin.firestore().collection('comments').doc(request.query.id).update({comment:request.body.data.comment}
).then(() => {
         response.send({"data": "Updated document in database"});
      });
   });
});
```

functions>index.js

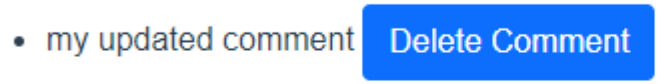
Test function

Passing in the updated comment data



Client code - update

- We want to provide a mechanism to edit comments.
- They are currently displayed as raw text values



Client code - update

- Using conditionals, when the user clicks on the text we'll pop in a textbox (input) with the current value and two buttons, one to save and one to cancel.

```
<div class="mb-3">
 <ul id="array-rendering">
  <li v-for="comment in comments">
   <div v-if="!editing">
    <span class='text' @click="enableEditing(comment.comment)">{{comment.comment}}</span>
   </div>
   <div v-if="editing">
    <input v-model="tempValue" class="input"/>
    <button @click="disableEditing"> Cancel </button>
    <button @click="save(comment.id)"> Save </button>
   </div>
   <button type="button" @click="deleteComment(comment.id)" class="btn btn-primary">Delete Comment</button>
  </li>
 </ul>
</div>
```

src>Blog.vue

Methods


```
enableEditing(comment){
  this.tempValue = comment;
  this.editing = true;
},
disableEditing(){
  this.tempValue = null;
  this.editing = false;
},
save(id) {
  const functions = getFunctions(app);
const updateComment = httpsCallable(functions,
'updatecomment?id='+id);
  updateComment({"comment":
this.tempValue}).then((result) => {
    this.getComments();
    this.editing = false;
  })

}
```

Sets the comment to a tempValue and sets the editing to true

Reverses it by clearing tempValue and setting editing to false

Call updatecomment passing in the ID. This is then passed to the function and then getComments is called to refresh the UI and editing is set to False

src>Blog.vue

Data function


```
data(){
  return {
    handle:'',
    comment:'',
    comments:[],
    tempValue:"",
    editing:false
  }
},
```

The data function will need to include additional variables for tempValue and editing

Exercise 2

- Implement both update and delete functionality.
- You should now have the full CRUD support.
- Modify your page to display the comments when the page loads.

FIREBASE EMULATORS – WORKING LOCALLY AND SAFELY

Dr. Enda Barrett

Overview

- In class - debugging in Web Applications
  - CORS errors
- Firebase emulation
  - Initialising the emulators
  - Installing Java for Firestore
- Invoking local functions instead of the deployed functions in the cloud

Firebase emulator

- Firebase provides an emulator which allows us to deploy our functions locally to test if they work as expected prior to deployment.
- Realistically we should only deploy once we are happy that our code is working as expected!
- Enter emulators!

Firebase emulator

- The emulator allows you to mimic the deployment environment on your laptop or PC where you can safely code features without worrying about breaking anything.

- Once you are happy you can deploy onto the cloud (Firebase) and users from around the globe can access your updates/changes to your site.

Emulating, hosting, functions and db

- Up until now when running the command ***npm run serve*** a local hosting server has been spun up, accessible on http://localhost:8080

- As you coded the frontend in real time, you were able to inspect your changes instantly without having to wait until everything uploaded to the cloud.

- This dramatically speeds up development, but now we wish to include **firebase functions** and **firestore db** to the local dev environment. This means we can work much faster only deploying when we are happy with the build.

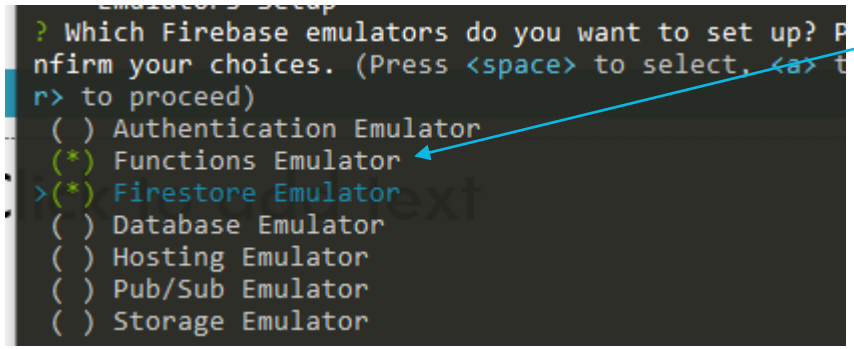
Initialise the emulators

- Firstly, we want to initialise the emulators we need
  - We only need to additionally include Firestore for the moment. Functions is automatically created (or at least it was on my setup).

- Open up a cmd/terminal window and enter the following command at the root folder of your project, i.e. hello-world
  - *firebase init emulators*

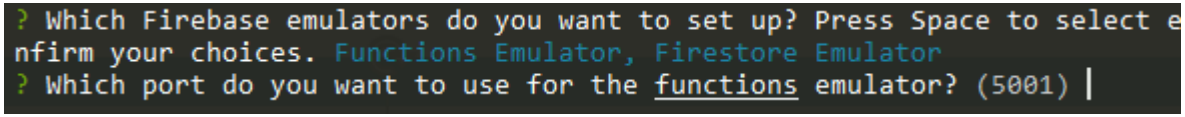
Select Firestore Emulator

- Choose Firestore Emulator



Functions emulator should already be installed

- Hit Enter to choose the default 5001 port for functions



Choose a different port Firestore

- The next step is to choose the port for Firestore, the default is 8080 but this is also the default port for many web servers and can conflict.
  - Change the port to **5002** and hit enter

```
=== Emulators Setup
? Which Firebase emulators do you want to set up? Press Space to select em
nfirm your choices. Functions Emulator, Firestore Emulator
? Which port do you want to use for the functions emulator? 5001
? Which port do you want to use for the firestore emulator? (8080) 5002
```

- Select Y to enable to emulator UI and choose any available port

```
=== Emulators Setup
? Which Firebase emulators do you want to set up? Press Space to select emulators, then Enter to confirm your choices. Functions Emulator, Firestore Emulator, Hosting Emulator
? Which port do you want to use for the functions emulator? 5001
? Which port do you want to use for the firestore emulator? 5002
? Which port do you want to use for the hosting emulator? 5000
? Would you like to enable the Emulator UI? (Y/n)
```

Download the emulators

- Select Y to download the emulators

```
=== Emulators Setup
? Which Firebase emulators do you want to set up? Press Space to select emulators, then Enter to confirm your choices. Functions Emulator, Firestore Emulator, Hosting Emulator
? Which port do you want to use for the functions emulator? 5001
? Which port do you want to use for the firestore emulator? 5002
? Which port do you want to use for the hosting emulator? 5000
? Would you like to enable the Emulator UI? (Y/n)
```

- This will download the jar file. The emulator platform is written in Java.

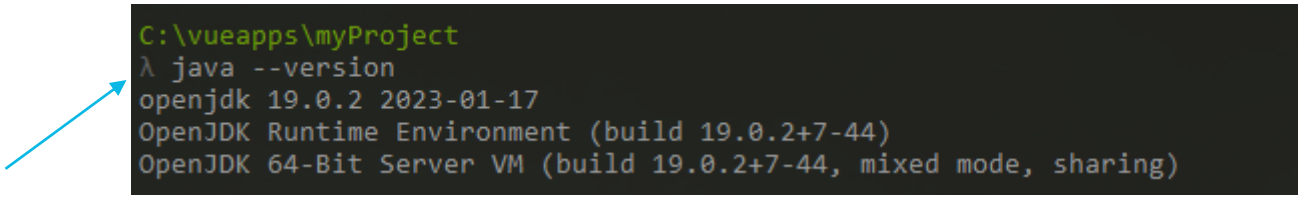
```
=== Emulators Setup
? Which Firebase emulators do you want to set up? Press Space to select emulators, then Enter to confirm y
? Which port do you want to use for the functions emulator? 5001
? Which port do you want to use for the firestore emulator? 5002
? Which port do you want to use for the hosting emulator? 5000
? Would you like to enable the Emulator UI? Yes
? Which port do you want to use for the Emulator UI (leave empty to use any available port)?
? Would you like to download the emulators now? (Y/n) Y
```

Java must be installed

- The firestore emulator runs on Java. Since all of you are doing java most of you should have java already installed, if not then please get it
  - https://openjdk.java.net/install/

- You can test that Java is installed by executing the command
  - java --version

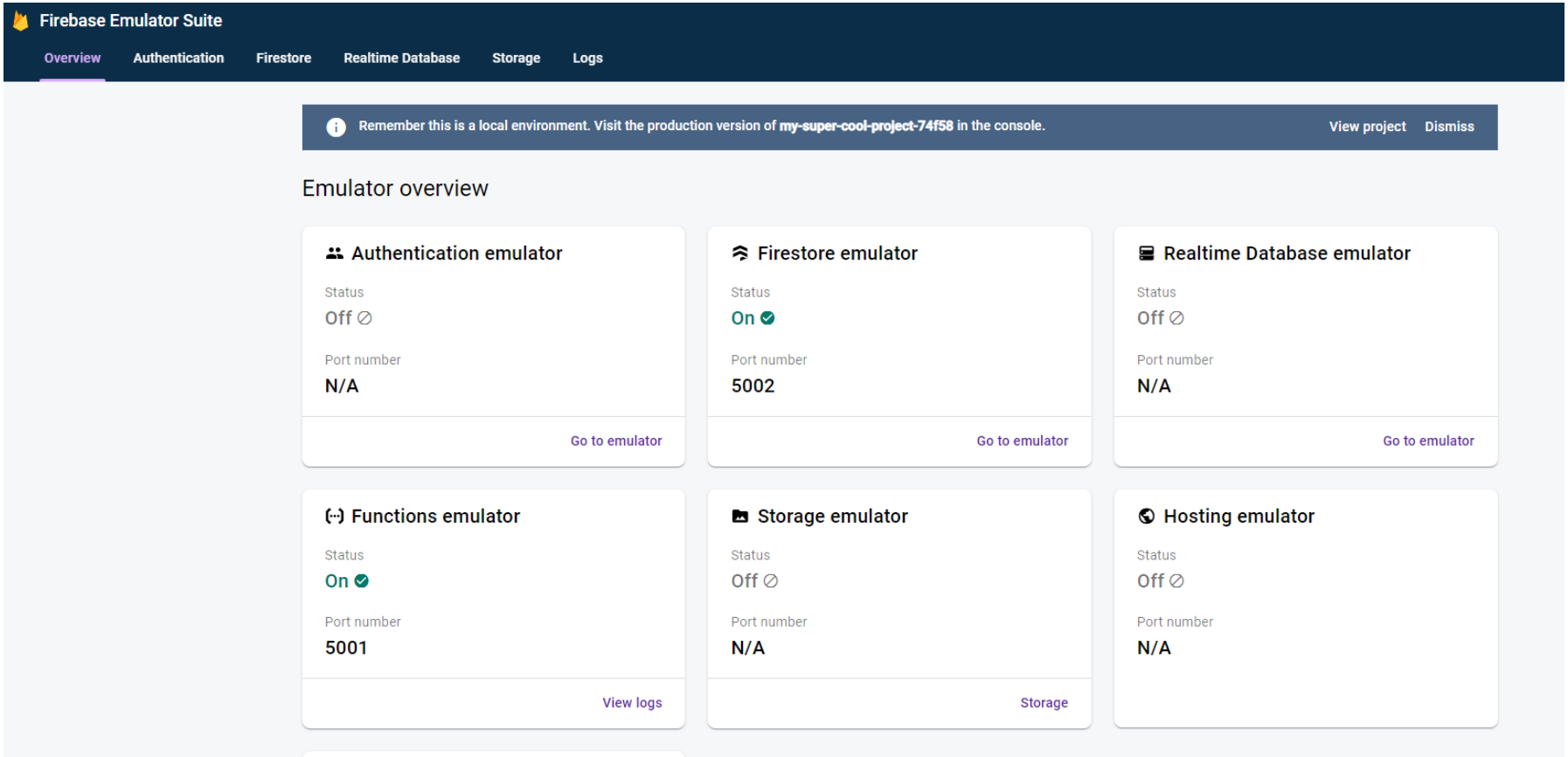
You should see this if Java is installed



- Note that the npm run command just opens up the package.json command and searches for a serve property and substitutes whatever command it finds and executes it.

- In our case that is simply **firebase emulators:start**

localhost:4000



Summary

- Firebase emulation is a great way to code safely offline.
- It's much quicker, make a small change in your functions they are instantly deployed
- Once you are happy with your fixes, features offline then you can push it to production, by deploying it.

Exercise 1 – Get the local env working

- First challenge is to get the local dev environment working properly
  - Set up emulation for Firestore
  - If Functions is not set up then set it up
  - You don't need to set up hosting because we will be using the default VueJS bootstrapped server to run our apps via npm run serve

Calling our functions - localhost

- If you examine the VusJS client code, we currently have a "httpsCallable" defined to allow us to post comments.
- We'd like to work locally, instead of calling the cloud deployed functions "https://us-central1-my-super-cool-project-74f58.cloudfunctions.net/postcomment" instead we'd like to invoke "http://localhost:5001/my-super-cool-project-74f58/us-central1/postcomment"

Recap: Blog.vue from 3 weeks ago


```
<script>
import app from '../api/firebase';
import { getFunctions, httpsCallable } from "firebase/functions";

export default {
 name: "Blog",
 data() {
  return {
   handle: '',
   comment: ''
  }
 },
 methods : {
  postComment () {
   console.log(this.handle);
   console.log(this.comment);
   const functions = getFunctions(app);
   const postComment = httpsCallable(functions, 'postcomment');
   postComment({"handle": this.handle, "comment":
this.comment})
      .then((result) => {
       // Read result of the Cloud Function.
       /** @type {any} */
       console.log(result);
      });
  }
  }
}
</script>
```

Import the exported app from api/firebase with the app config.

Import the objects getFunctions and httpsCallable from firebase/functions

Pass in the app config as a param to getFunctions

Using httpsCallable, pass in functions and the name of the function you wish to call

Pass the object as an argument to postComment

Modify our VueJS code

- In order to hit our local functions we can simply do a location check within our code. If the current location of the client request is “localhost” then use the local function emulators, if it’s not, i.e. in production (on Firebase) then use the Firebase functions and save the data to the remote Firestore database instead of the local one.
- document.window.hostname in JS will give us the host name from the URL

Blog.vue


```
import app from '../api/firebase';
import { getFunctions, httpsCallable, connectFunctionsEmulator } from
"firebase/functions";

export default {
  name: "Blog",
  data(){
    return {
      handle:'',
      comment:''
    }
  },
  methods: {
    postComment(){
      console.log(this.handle);
      console.log(this.comment);

      const functions = getFunctions(app);
      if(window.location.hostname === 'localhost') // Check if working locally
        connectFunctionsEmulator(functions, "localhost", 5001);
      const postComment = httpsCallable(functions, 'postcomment');
      postComment({"handle": this.handle, "comment": this.comment}).then((result) => {
        // Read result of the Cloud Function.
        // /** @type {any} */
        console.log(result);
      });
    }
  }
}
```

Import connectFunctionsEmulator

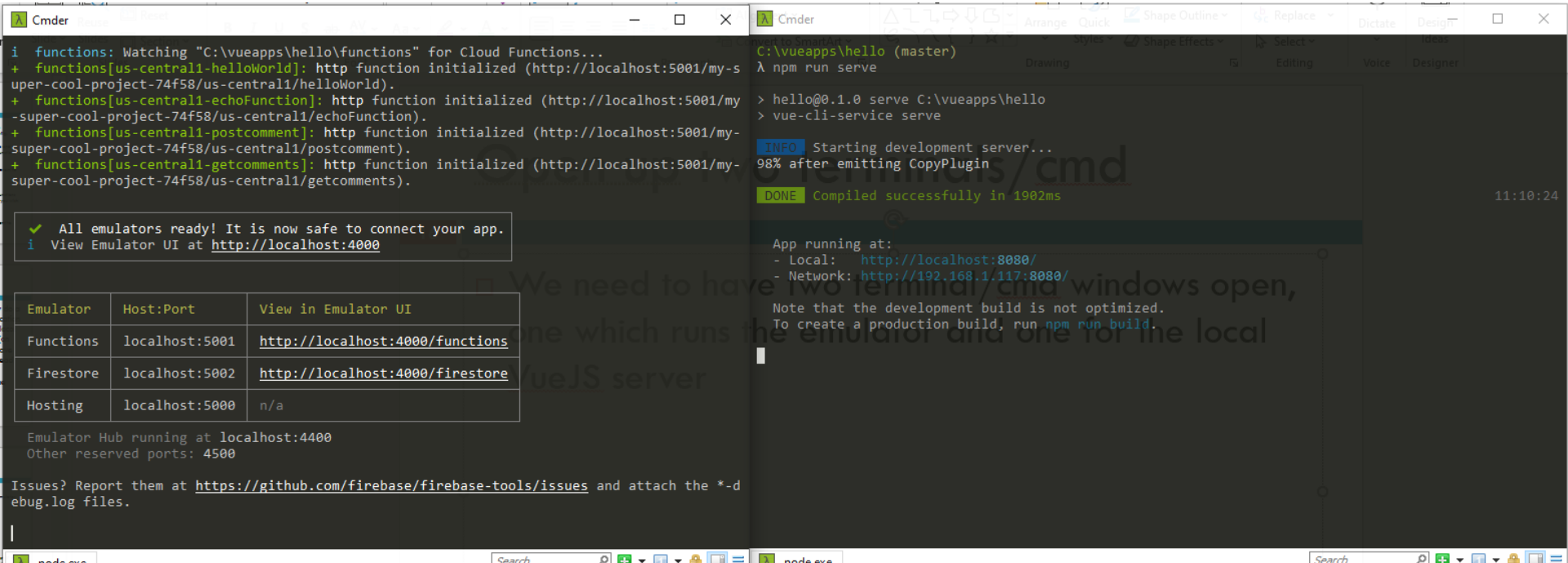
Check for localhost, if it is then connect to the local function emulator

Open up two terminals/cmd

- We need to have two terminal/cmd windows open, one which runs the emulator and one for the local VueJS server

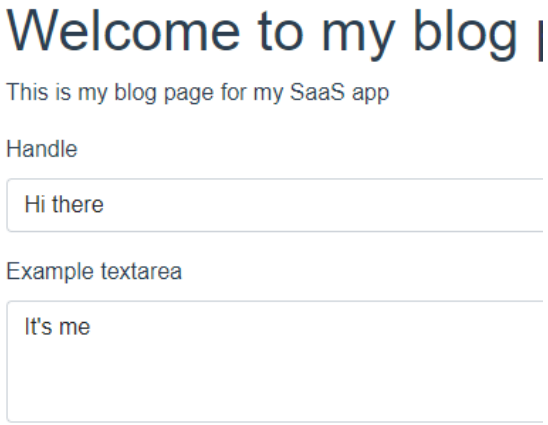
firebase emulators:start

npm run serve



Test it locally

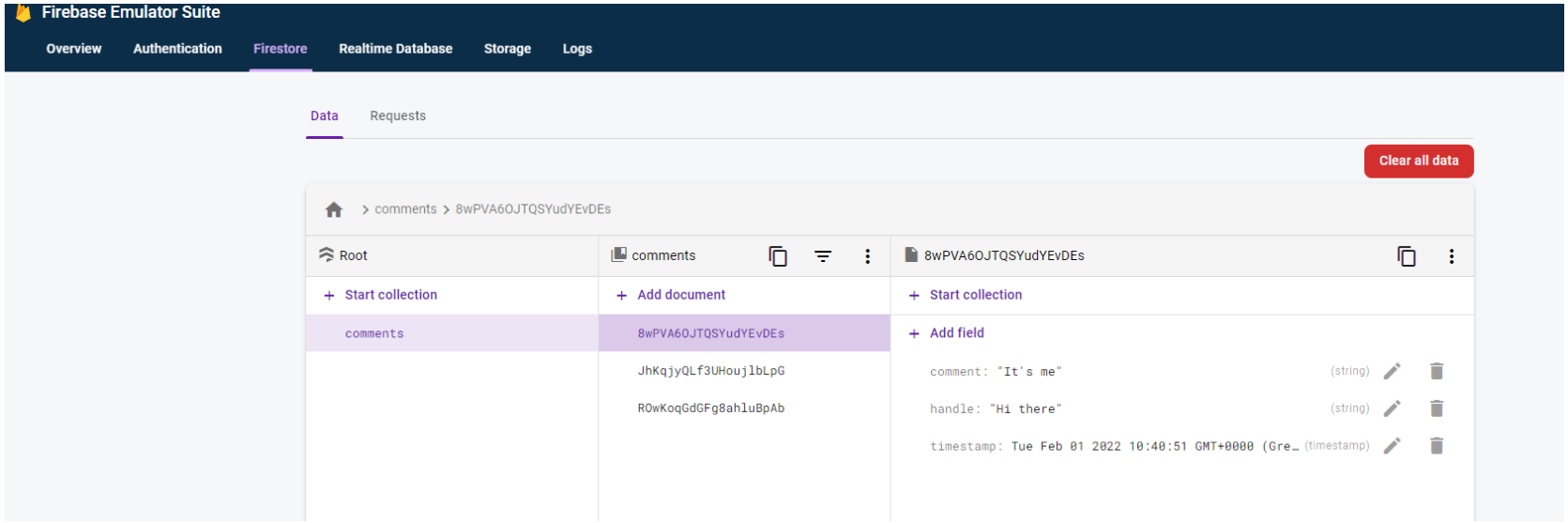
- If you try and post a comment from the /blog page



- Local functions will be invoked and the data saved in a Firestore database (see next slide)

Firestore DB emulator

- The data should now be visible only in the local db http://localhost:4000/firestore



Be careful!

- The database is ephemeral – once you shutdown the emulators, i.e. cancel it using Ctrl + C on the cmd window, the database will be purged.

- If you are working on functions that read data from the data (such as getcomments) then make sure there is data present within the database.

Deploy and Test

- Execute the command npm run build
  - This will create a production build

- Once complete, execute the command firebase deploy

- When you navigate to the cloud url i.e. https://my-super-cool-project-askf.web.app/ it should work and the data you post via the comment form should save in the Cloud Firestore database and not the local one

Exercise

- Modify your serverless functions to work with local emulation
- Test it works both locally and when deployed on the cloud

SOFTWARE TESTING THEORY– CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

Lecture overview

- Software Testing
  - Importance of testing
- Testing techniques
  - Black box testing
  - White box testing
- Code coverage
- Integration Testing
- Test Automation

Importance of testing

- Adding tests to your application code serves two purposes
  - Ensure that the code you have written behaves as expected
  - Ensure that future changes by you or another member of your team don't result in an undesired output

- Testing in NodeJS adds a little more complexity as you need to be able to handle, synchronous, asynchronous and RESTful functionality

Why do we test software?

- To ensure the delivered software is high quality we must test frequently
- Quality control is about finding variations in a work or end product compared with a given specification
- The purpose of quality control is to subject the software to frequent tests in order to eliminate problems as early as possible.

Testing techniques

- There are many possible testing techniques which can be intermingled to provide the best possible testing coverage.
- No matter how extensive the testing is, it cannot be exhaustive and guarantee program correctness.
- There will always be bugs!

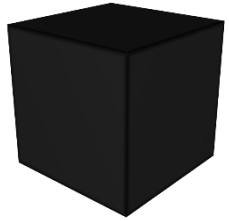
Lecture overview

- ~~Software Testing~~
  - ~~Importance of testing~~
- **Testing techniques**
  - **Black box testing**
  - **White box testing**
- Code coverage
- Integration Testing
- Test Automation

At a high level : Black box testing

- Assumption that the tester doesn't know the internal workings of the program under test.
- The unit is considered a black box whereby certain inputs are provided and specific outputs are expected
- Also known as testing to specs.

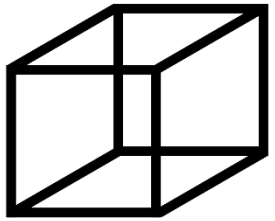
Black box testing cont.



- Because no implementation knowledge is needed this form of testing can be completed by the users themselves.
- This is the main technique for acceptance testing
- Black box testing tends to search for missing functionality, i.e. an expected functional requirement has not been implemented in the software

White box testing (Glass box)

- As you can imagine white box testing is the opposite of black box testing.



- The program under test is considered fully observable i.e. a transparent glass box that one can see inside.

- The tester must study the code and define tests that exercise selected execution paths in the code.

Lecture overview

- ~~Software Testing~~
  - ~~Importance of testing~~
- ~~Testing techniques~~
  - ~~Black box testing~~
  - ~~White box testing~~
- **Code coverage**
- Integration Testing
- Test Automation

Code coverage

- What is it?
  - You hear a lot about code coverage these days, it's essentially the amount of your code that is "covered" by a test
  - As a metric it can help you determine what % of code is not covered
  - Usually aim for 80-90%
  - Generally two main types:
    - Statement/line coverage – is each executable line of code covered by a test?
    - Branch coverage – is each conditional branch covered by a test
- In JavaScript one of the most popular is Istanbul
  - https://istanbul.js.org/

Statement/Line coverage

- Statement or line coverage is simply a measure of the number of lines of your codebase that are "covered" by a test.
- If we were testing a particular function such as the retirement calculator, a test that invoked the function with a valid age parameter such as 40, would ensure 100% code coverage

```
function retire(age)
{
   var added = 68-age;
   return 2020+added;
}
```

Branch (conditional) coverage

- Let's assume that the government allows those currently over 40 to retire at 65 and those under 40 must work until 68. Our retirement calculator now looks like this

```
function retire(age)
{
   var added = 0;
   if(age > 40)
      added = 65-age;
   else
      added = 68-age;

   return 2020+added;
}
```

- In order to have 100% coverage, we will need two tests one where the age is 40+ and one where the age is less than 40. If we only have a single test for either case, then the coverage will only be 50%.

Lecture overview

- ~~Software Testing~~
  - ~~Importance of testing~~
- ~~Testing techniques~~
  - ~~Black box testing~~
  - ~~White box testing~~
- ~~Code coverage~~
- **Integration Testing**
- Test Automation

Naïve test : Testing with Postman

- Throughout our development effort on the backend most of us have experimented with the Postman client to ensure that our backend APIs operate as they are supposed to.
- This testing however was most likely naïve, passing in expected parameters and observing expected outcomes
- What about testing
  - A POST request with different/unexpected data in the body
  - A delete request with an unusual id or no id?
  - A get request with no id…

Postman relies on manual testing

- Remember that each request you create in Postman has to be manually run each time. It will prove difficult to remember to run each test for every REST endpoint.
- A moderately sized application such as an ecommerce site could easily contain hundreds of REST endpoints, handling user data, shopping carts, wishlists, shipping details, order details…
- Do you want to examine the results of 100s of requests each time you wish to test the system?

Integration Testing

- An integration test is performed to make sure that different bits of the application work together. Making a HTTP request to a REST API such as /getComments, involves a database call. If authentication is also required to view comments then that could also be verified.
- A well designed integration test will help the developer to keep track of changes to the system.
- It's a more convincing test, to ensure that the system behaves as expected.

Integration testing contd.

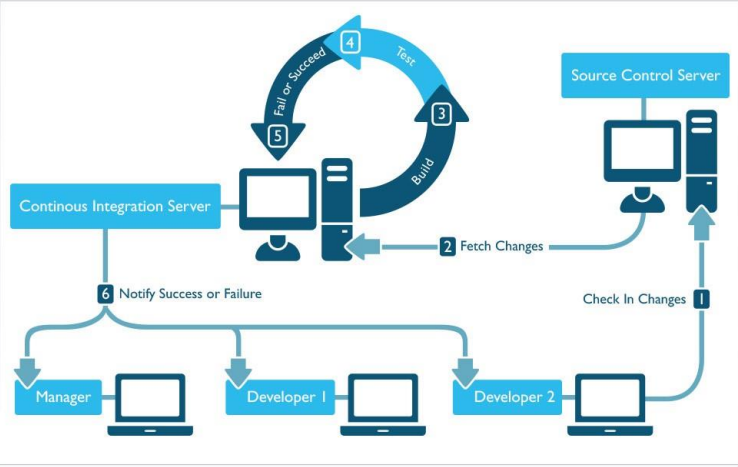
- In reality integration testing is a logical extension of unit testing.
- It tests the interfaces between units
- When you combine/integrate multiple units, the interfaces/boundaries between these become crucial pain points in the wider software development effort.

Lecture overview

- ~~Software Testing~~
  - ~~Importance of testing~~
- ~~Testing techniques~~
  - ~~Black box testing~~
  - ~~White box testing~~
- ~~Code coverage~~
- ~~Integration Testing~~
- **Test Automation**

Test Automation

- What is it?
  - During the build process, pre-scripted automated tests are executed to ensure that quality is maintained prior to deployment
- Usually this happens when you commit your code to source/version control. A Continuous Integration server generates a build.



Summary

- Software Testing
  - Importance of testing
- Testing techniques
  - Black box testing
  - White box testing
- Code coverage
- Integration Testing
- Test Automation

TESTING OUR CODE – CT 216 SOFTWARE ENGINEERING I

Dr. Enda Barrett

Lecture overview

- Mocha (a JS test framework)
  - Installation
- Testing our (backend) Firebase functions
- Chai assertion library
- Creating our first test
- Running our tests

Mocha

- Mocha is one of the more popular testing frameworks for JavaScript, allowing you to test backend NodeJS code as well as client code in the browser.

- Mocha will allow us to create a suite of tests, describe the function of each test, generate reports etc

https://mochajs.org/

Mocha terms

- *describe*
  - describe method creates a suite of test cases
  - it's merely for grouping tests only, allows you to give a high level description
  - You can next them also, i.e. groups within groups
- *it*
  - it method implements a specific test case

Install mocha

- Probably best to install mocha globally on your machine because we'll be testing both client and server side

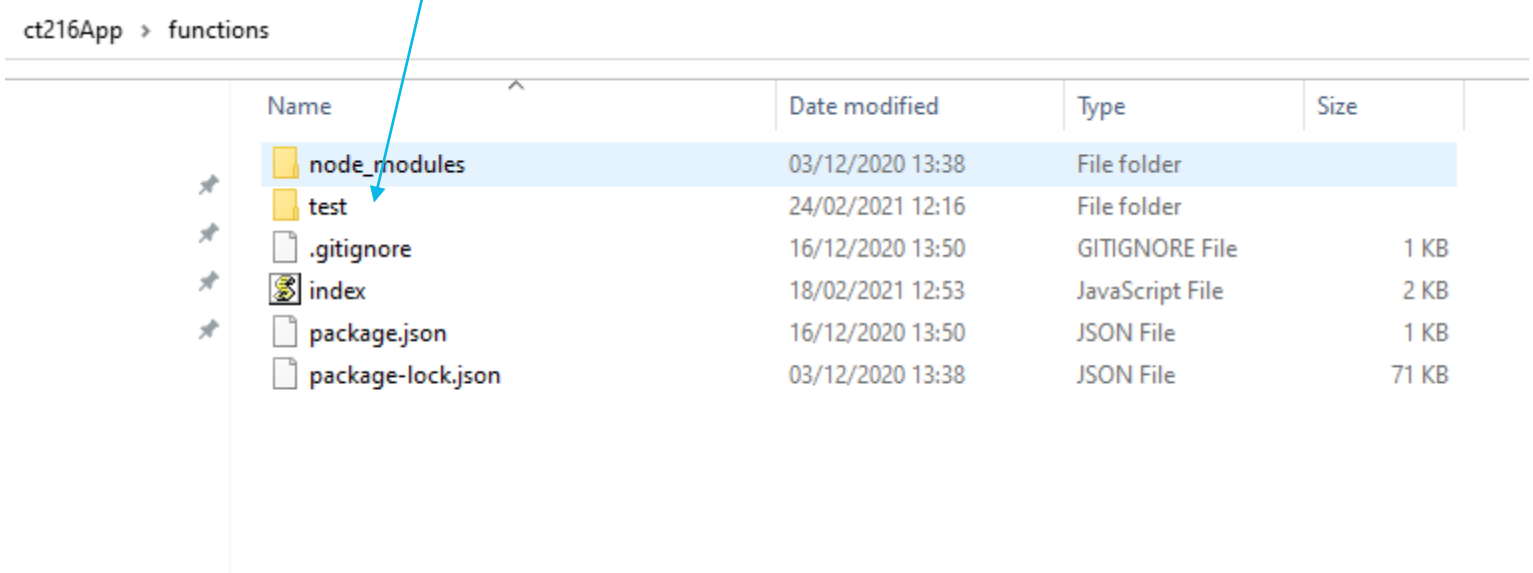
- npm install -g mocha

Note: Remember to put sudo in front of the command if running on a mac

Testing our functions - Add a test folder

- Let us start with testing our functions

- Create a test folder inside the **functions** folder

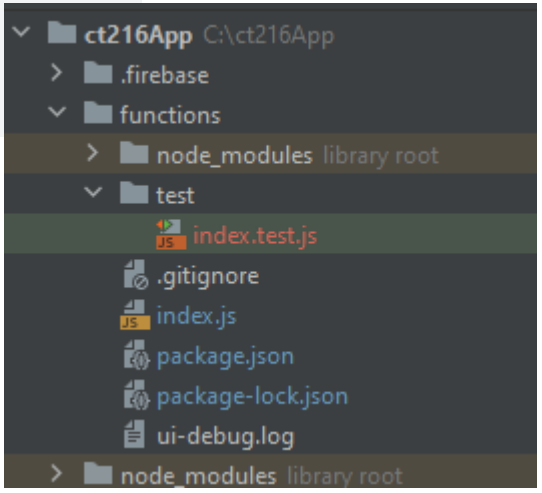


Create a new file and add it to test

- Create a new file inside in the functions folder and call it *index.test.js*

- Modify the package.json file located under the functions folder and add the following

```
"scripts": {
  "test": "mocha --reporter spec"
}
```



Creating our first test

- At this point we haven't created any specific tests yet, so lets now create one.

- The premise is that our functions are HTTP functions, i.e. you invoke them through HTTP calls and they respond. If you send in some test data and they respond with the correct response you know they are working as they should be

Chai assertion library

- In our tests, we would like to be able to check or assert that certain conditions are upheld or the request body contains certain content.

- Chai is an assertion library, which allows for speedy checking of our response data

- Note: Check the API docs online for the full suite of assertions available

http://chaijs.com/

Testing HTTP Functions

- The functions written to date, such as getcomments or postcomments, are invoked through HTTP requests. We would like write tests to ensure that no poor developer breaks our wonderful code!

- *npm install chai*
- *npm install chai-http*

Unit Testing

- One of the very first tests often encountered is the unit test.
- A unit test is written by the programmer to verify that a small piece of functionality behaves as it should.
- It is usually narrow in scope, should be easy to write and run.
- Heavy reliance on mocking data is often associated with unit testing.
- A unit test should not have dependencies outside the unit, i.e. connection to a data, call to a 3rd party API, should not form part of a unit test.

index.test.js

- Create test that writes a comment to the db

Function address

Unit Test

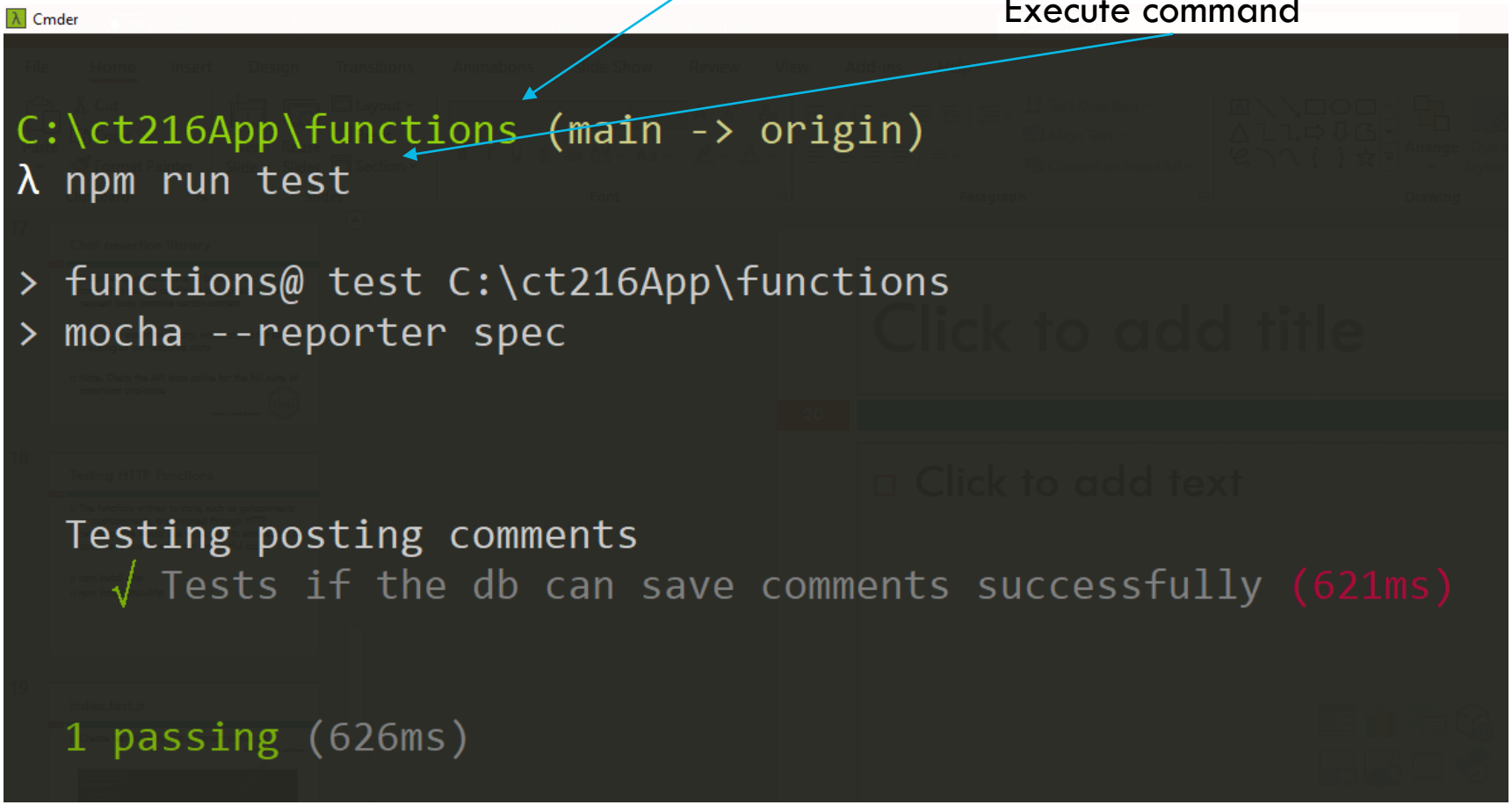
```
const chai = require('chai');
const chaiHttp = require('chai-http');

chai.use(chaiHttp);
const expect = chai.expect;
describe('Testing posting comments', function () {
    this.timeout(100000);
    it('Tests if the db can save comments successfully', async () => {
        const result = await chai.request('https://us-central1-my-awesome-project-86da3.cloudfunctions.net')
            .post('/postcomment')
            .set('content-type', 'application/json')
            .send({ data : {handle: 'TestEnda', comment: 'Test comment from Enda'} } );
    });
});
```

functions/test/index.test.js

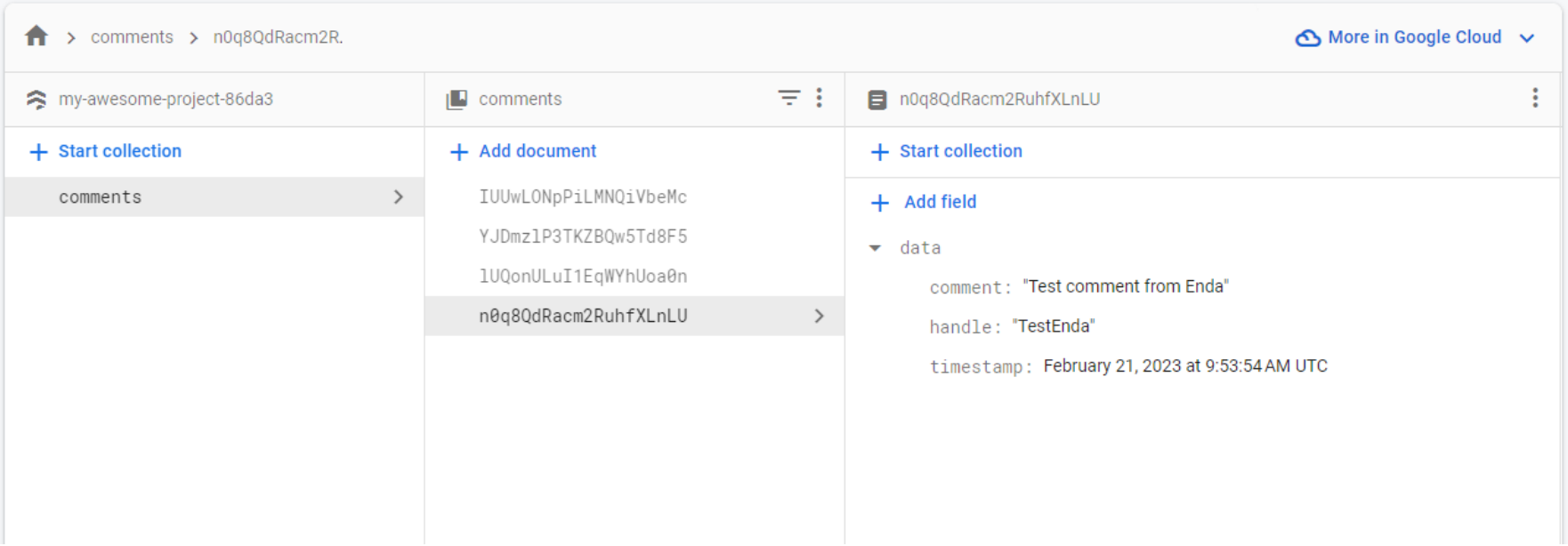
Run our first test

Change to Functions Directory

Execute command



Check database



index.test.js

- Add additional test to index.test.js

```
const chai = require('chai');
const chaiHttp = require('chai-http');

chai.use(chaiHttp);
const expect = chai.expect;
describe('Testing posting comments', function () {
    this.timeout(100000);
    it('Tests if the db can save comments successfully', async () => {
        const result = await chai.request('https://us-central1-my-awesome-project-86da3.cloudfunctions.net')
            .post('/postcomment')
            .set('content-type', 'application/json')
            .send({ data : {handle: 'TestEnda', comment: 'Test comment from Enda'} } );
    });
});

describe('Tests Get Comments', function () {
    this.timeout(100000);
    it('Tests if there are comments', async () => {
        const result = await chai.request('https://us-central1-my-awesome-project-
86da3.cloudfunctions.net').get('/getcomments');
        expect(result.statusCode).to.equal(200);
        expect(result.body.data).to.be.an('Array');
    });
});
```

Run our tests again


```
C:\ct216App\functions (main -> origin)
λ npm run test

> functions@ test C:\ct216App\functions
> mocha --reporter spec


  Testing posting comments
    √ Tests if the db can save comments successfully (632ms)

  Tests Get Comments
    √ Tests if there are comments (609ms)


  2 passing (1s)
```

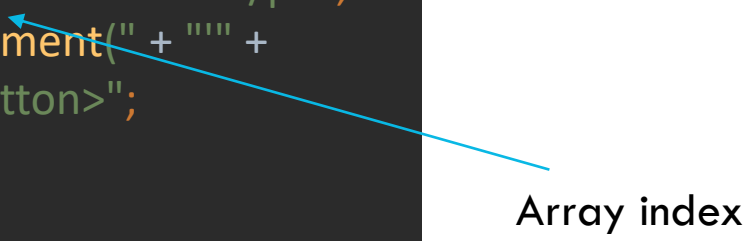
More important assertions

- Whilst checking for MIME types, HTTP response codes and other header related content is important, these alone whilst possibly causing some unexpected behaviour may not break your system.
- Changes to field names, missing data or expected params can have detrimental effects on your application.
- Another colleague (or you!) may have made these modifications, and it would be nice to ensure that any modifications to the schema, or invalid data will throw an error

This is important

If the response was modified to return a string instead of an array, it would break our client code

```
for (var i = 0; i < data.length; i++) {
   sHTML += "<p> Handle: " + data[i].handle + "</p>";
   sHTML += "<p> Comment: " + data[i].comment + "</p>";
   sHTML += "<button onclick=deletecomment(" + "'" +
data[i].id + "'" + ")>Delete Comment</button>";
   comments.innerHTML = sHTML;
}
```



Ensure a property exists

- We know that our client code will break if we don't have any **comment** field – so create a test!

```
describe('Tests Get Comments', function () {
    this.timeout(100000);
    it('Tests if there are comments', async () => {
        const result = await chai.request('https://us-central1-my-
awesome-project-86da3.cloudfunctions.net').get('/getcomments');
        expect(result.statusCode).to.equal(200);
        expect(result.body.data).to.be.an('Array');
        expect(result.body.data[0].data).haveOwnProperty('comment');
    });
});
```

Remember it's an array so you must query an object by index position

Lecture summary

- Mocha (a JS test framework)
  - Installation
- Testing our (backend) Firebase functions
- Chai assertion library
- Creating our first test
- Running our tests

FIREBASE AUTHENTICATION

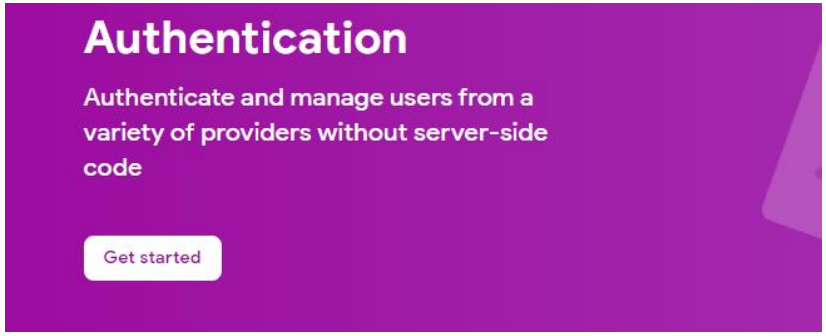
Dr. Enda Barrett

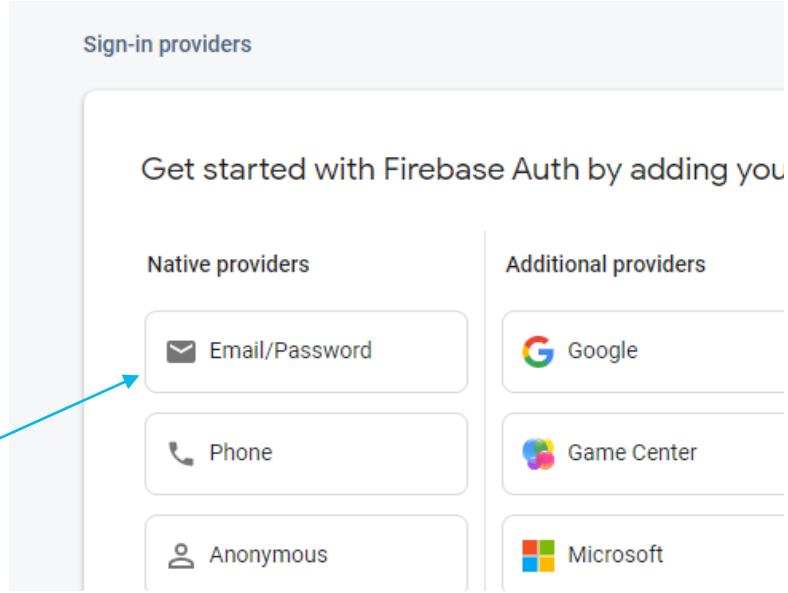
Lecture Overview

- Firebase Authentication Framework
- Creating separate pages (SFCs) for Registration and Login
- Using the Firebase Auth client libraries to register users and login
- Creating a Secure.vue SFC
  - Routing using Navigation Guards
- Creating a secure serverless function
- Theory on Authentication - Tokens, hashing,

Firebase Auth

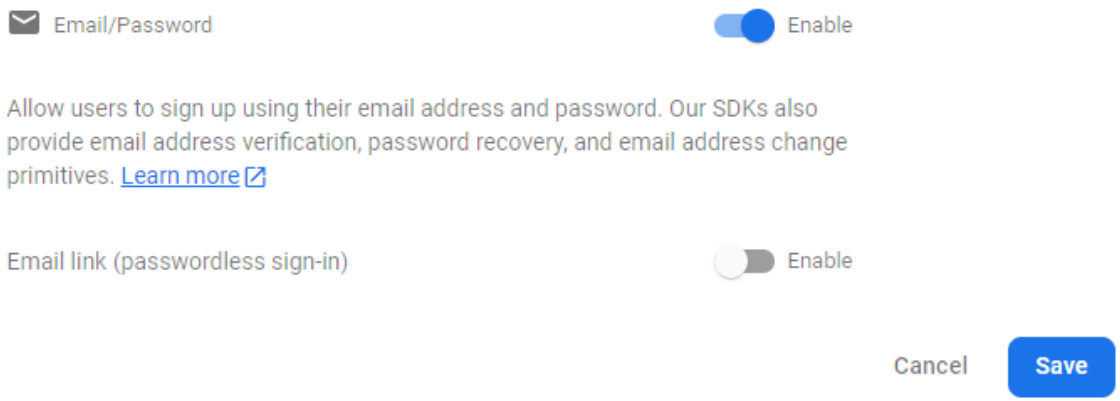
- Firebase provides an Auth framework which allows us to create users, login users for our apps.
- Login into the firebase console, click the Authentication tab and then select *Get Started*
- We are going to enable email/password authentication



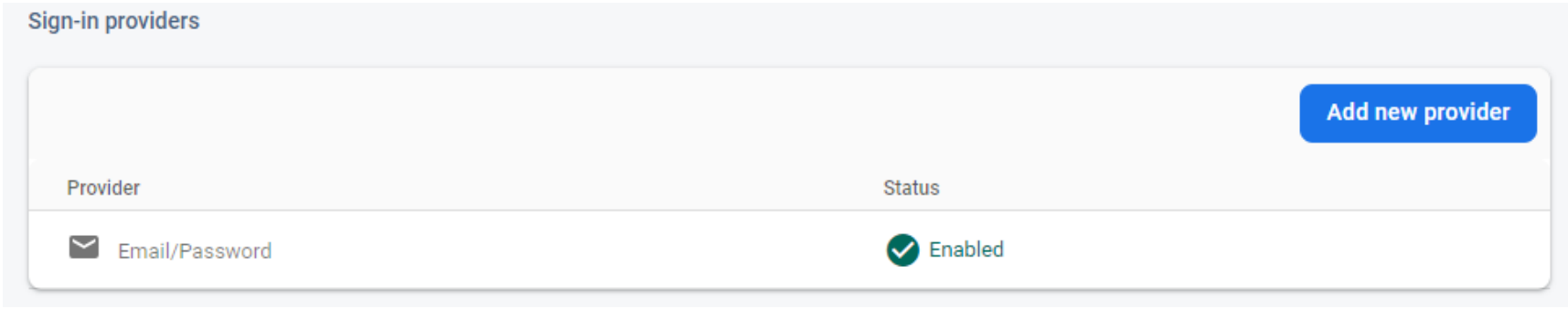


Enable email/password

- Click save



Successfully enabled you should see this



Firebase client libraries

- Firebase provides us with client JavaScript libraries that we can use to create accounts, log user's in etc.
  - Should already be installed in node_modules folder
- In order to use them on a webpage we must import them, the same we import any other JS library.
- Details of the auth framework and samples are available here
  - https://firebase.google.com/docs/auth

Create a registration page

- Add a SFC page (under the pages folder) to our apps called Registration.vue
- Pop in input fields for an email address and password and a button for Create Account
- See code on next slide for Registration.vue

```
<div class="container">
 <div class="form-group">
 <label for="exampleInputEmail1">Email address</label>
 <input type="email" v-model="email" class="form-control" id="exampleInputEmail1"
aria-describedby="emailHelp" placeholder="Enter email">
 <small id="emailHelp" class="form-text text-muted">We'll never share your email with
anyone else.</small></div>
 <div class="form-group">
  <label for="exampleInputPassword1">Password</label>
  <input type="password" v-model="password" class="form-control"
id="exampleInputPassword1" placeholder="Password">
 </div>
 <button @click="register" class="btn btn-primary">Create Account</button>
 </div>
```

2-way binding on email and password – v-model attributes

A register method must also be defined

components>Registration.vue

Add the firebase client libraries

- Next step is to add the Firebase client libraries to the register web page and the event handler for registration
- Instructions and docs available at this URL
  - https://firebase.google.com/docs/auth/web/start
- This is a great source for finding more information about the SDKs

```
<script>

import app from "../api/firebase"
import { getAuth, createUserWithEmailAndPassword } from "firebase/auth";

export default {
  name: "Registration",
  data(){
    return {
      email: "",
      password:""
    }
  },
  methods : {
    register(){
      const auth = getAuth(app);
      createUserWithEmailAndPassword(auth, this.email, this.password)
        .then((userCredential) => {
          // Signed in
          const user = userCredential.user;
          console.log(user)
          // ...
        })
        .catch((error) => {
          const errorCode = error.code;
          const errorMessage = error.message;

          console.log(errorCode)
          console.log(errorMessage)
          // ..
        });
    }
  }
}

</script>
```

Import the config details from api/firebase as well as the objs from firebase/auth

Create a register method. Pass in the app init as a parameter to the getAuth constructor.

Pass the returned “auth” object, with the email and password to the createUserWithEmailAndPassword method which will create a user on the app.

Log everything to the console initially to test if everything works

src/pages/Register.vue

Add a route to the Registration SFC

- In order to render the page we need to route to it.

```
function loadPage (component) {
   // '@' is aliased to src/components
    return () => import(/* webpackChunkName: "[request]" */ `@/pages/${component}.vue`)
    }
    export default [
       { path: '/', component: loadPage('HelloWorld')},
       { path: '/blog', component: loadPage('Blog') },
       { path: '/about-us', component: loadPage('AboutUs')},
       { path: '/registration', component: loadPage('Registration')}
    ]
```

router>routes.js

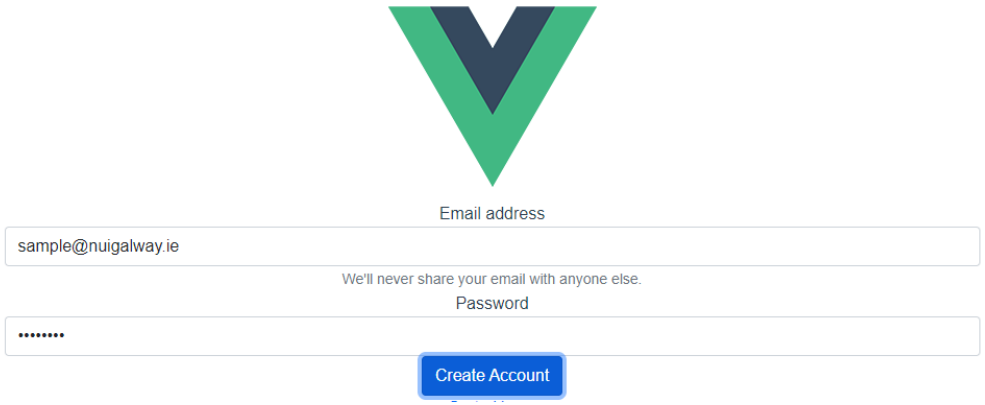
- Pop a link to the page on the nav

```
<template>
<router-link to="/">Home</router-link>
<router-link to="/blog">Blog</router-link>
<router-link to="/registration">Sign Up</router-link>
</template>
```

components>Navigation.vue

Open up a browser and create a user

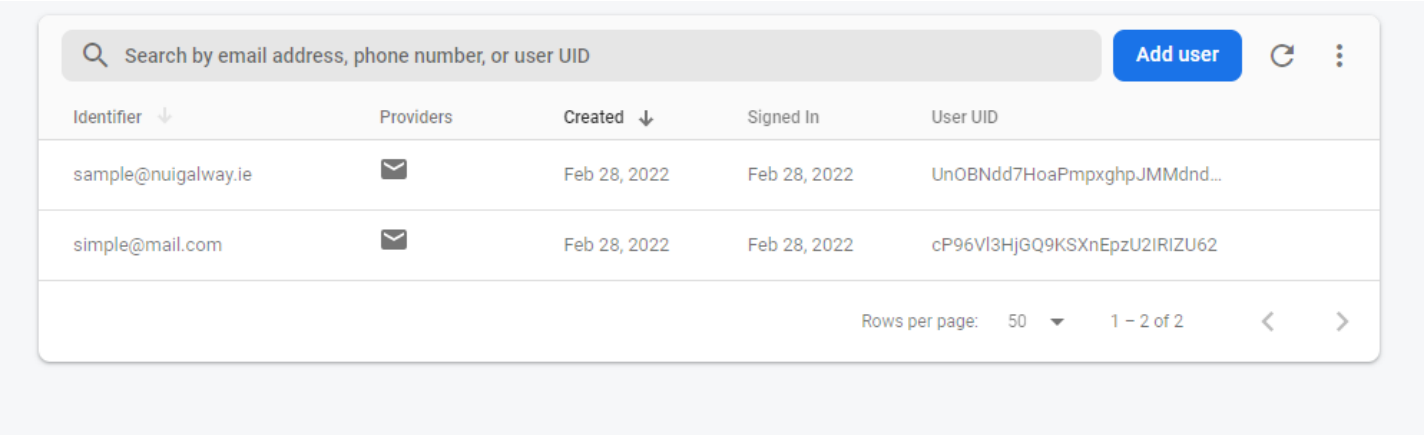
- Now it's time to test whether the account creation component that we have just built actually works



Run the app and create a new user

- You should see the user listed in the browser console and you should also be able to see it listed on Firebase under Authentication

- Well done you've created your first user



Exercise 1

- Create a registration page in your apps using the code from slide 7
- Using the code in Slide 9, create your very first user account, that users will be able to use to login
- Include a route to your new page

Login page

- The next step is to create a login page to allow users who are registered to login using their accounts
- The login page is just going to be largely the same as the Registration.vue page, with an email and password field and also a login button
- Add a new SFC under the pages folder called Login.vue and pop in the HTML on the next slide

Login.vue is similar to Registration.vue


```
<template>
  <div class="container">
    <div class="form-group">
      <label for="exampleInputEmail1">Email address</label>
      <input type="email" v-model="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="Enter email">
      <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small></div>
    <div class="form-group">
      <label for="exampleInputPassword1">Password</label>
      <input type="password" v-model="password" class="form-control" id="exampleInputPassword1" placeholder="Password">
    </div>
    <button @click="login" class="btn btn-primary">Login</button>
  </div>
</template>
<script>
export default {
  name: "Login",
  data(){
    return {
      email: "",
      password:""
    }
  },
  methods : {
    login(){

    }
  }
}
</script>
<style scoped>
</style>
```

Add the js to Login.vue


```
<script>
import app from "../api/firebase"
import { getAuth, signInWithEmailAndPassword } from "firebase/auth";

export default {
  name: "Login",
  data(){
    return {
      email: "",
      password:""
    }
  },
  methods : {
    login(){
      const auth = getAuth(app);
      signInWithEmailAndPassword(auth, this.email, this.password).then((userCredential) => {
        // Signed in
        let user = userCredential.user;
        console.log(user);

      }).catch((error) => {
        let errorCode = error.code;
        let errorMessage = error.message;
        console.log(errorCode)
        console.log(errorMessage)
        });
      }
  }
}
</script>
```

pages>Login.vue

https://firebase.google.com/docs/auth/web/password-auth

Add a route to the Login SFC

- In order to render the page we need to route to it.

```
function loadPage (component) {
   // '@' is aliased to src/components
    return () => import(/* webpackChunkName: "[request]" */ `@/pages/${component}.vue`)
    }
    export default [
       { path: '/', component: loadPage('HelloWorld')},
       { path: '/blog', component: loadPage('Blog') },
       { path: '/about-us', component: loadPage('AboutUs')},
       { path: '/registration', component: loadPage('Registration')},
       { path: '/login', component: loadPage('Login')}

    ]
```

- Pop a link to the page on the nav

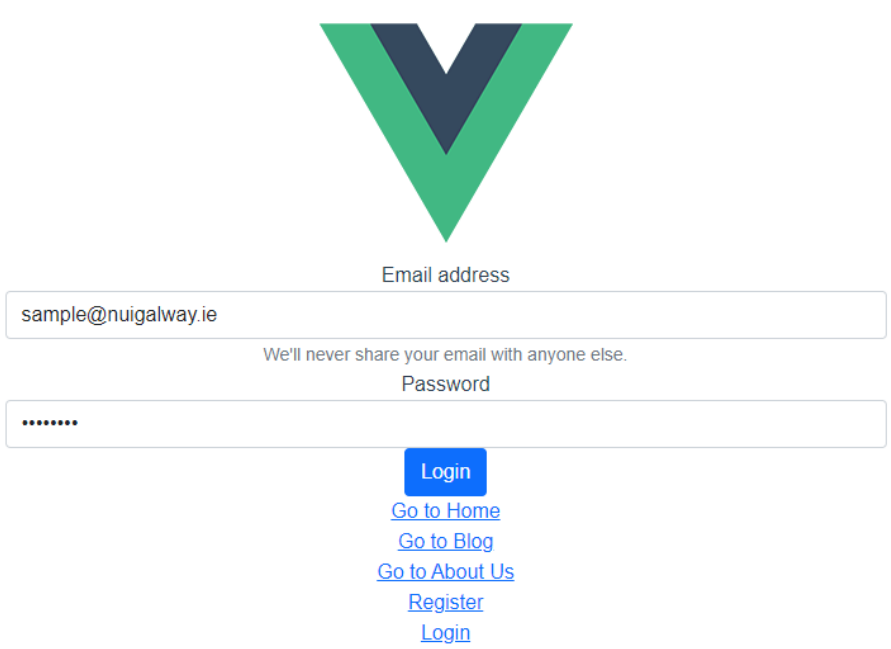
router>routes.js

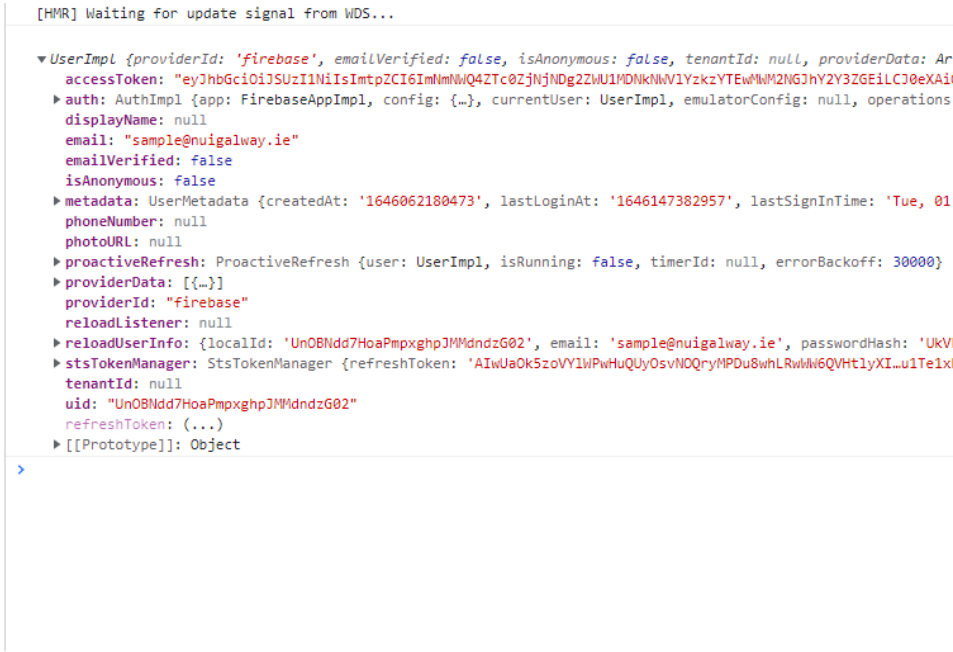
```
<template>
<router-link to="/">Home</router-link>
<router-link to="/blog">Blog</router-link>
<router-link to="/registration">Sign Up</router-link>
<router-link to="/login">Sign In</router-link>
</template>
```

components>Navigation.vue

Test an account

Using an existing account





Exercise 2

- Add a login page using the code from slides 15 and 16.
- Take an existing account that you have already created and attempt to log in.
- You should see in the browser console the user account returned by the auth framework

Securing our apps

- Securing access to individual pages (only an authenticated user can access a page)
- Securing access to serverless functions (only an authenticated user can invoke the function)
- Updating the UI based on Login status

Adding a secure section of the site

- Once your users have registered for the first time or if they have just logged in, you would like to direct users to a secure section of the web page

- The first thing is to add a new SFC called ***Secure.vue*** which will be our page that only logged in users can access

- Add a route to it ***routes.js***

Programmatic Navigation

- VueJS provides a method to navigate programmatically to pages via the JS code
- When the user has logged in successfully we can use this to load up the secure page

```
methods: {
  login() {
    const auth = getAuth(app);
    signInWithEmailAndPassword(auth, this.email, this.password).then((userCredential) => {
      // Signed in
      let user = userCredential.user;
      console.log(user);
      this.$router.push({path: '/secure'})
    }).catch((error) => {
      let errorCode = error.code;
      let errorMessage = error.message;
      console.log(errorCode)
      console.log(errorMessage)
    });
  }
```

https://router.vuejs.org/guide/essentials/navigation.html

Page still not secure

- All users can still access the page /**secure** whether they are logged in or not because access is not restricted just yet.

- The next step is to add a navigation guard to protect the route and only load the page for logged in users

https://router.vuejs.org/guide/advanced/navigation-guards.html

Routes.js


```
import { getAuth, onAuthStateChanged } from "firebase/auth";
import app from "../api/firebase";
function isAuth(to, from, next){
    console.log("Checking auth");
    const auth = getAuth(app);
    onAuthStateChanged(auth, (user) => {
        if (user) {
            console.log(user);
            // User is signed in so continue to desired page
            return next();
            // ...
        } else {
            // User is signed out
            // Send them back to the home page or maybe the login
page
            return next({path : '/'});
        }
    });
}
```

src>router>routes.js

Add the following to routes.js

onAuthStateChanged gets fired everytime a user logs in or out. By listening to this event we can check the login status on the user.

The **isAuth** function is a navigation guard, **to** provides details on the path the user is going to, **from** is where they came from and **next** is optional and provides a method called to continue navigation

Routes.js cont.


```
function loadPage (component) {
    // '@' is aliased to src/components
    return () => import(/* webpackChunkName: "[request]" */ `@/pages/${component}.vue`)}

export default [
    { path: '/', component: loadPage('Home') },
    { path: '/blog', component: loadPage('Blog') },
    { path: '/registration', component: loadPage('Registration') },
    { path: '/login', component: loadPage('Login') },
    { path: '/secure', component: loadPage('Secure'), beforeEnter: isAuth }
]
```

Before the page is entered check that the user is logged in!

src>router>routes.js

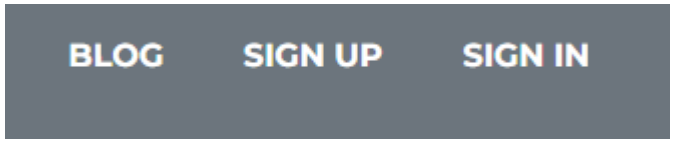
Logout

- It's important to enable users to log out of your application.
- The good news is that the framework provides a mechanism to handle this.
- We will place a logout link in the Navigation SFC

```
signOut().then(() => {
  // Sign-out
successful.
}).catch((error) => {
  // An error happened.
});
```

Navigation

- It's good to place the logout code in the Navigation SFC
- This allows us to show certain links such as Sign Up or Login if the user is not logged in or Logout if they are.



Navigation.vue


```
<template>
  <!-- Navigation-->
  <nav class="navbar navbar-expand-lg bg-secondary text-uppercase fixed-top" id="mainNav">
    <div class="container">
      <router-link class="navbar-brand" to="/">My Application</router-link>
      <button class="navbar-toggler text-uppercase font-weight-bold bg-primary text-white rounded" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">
        Menu
        <i class="fas fa-bars"></i>
      </button>
      <div class="collapse navbar-collapse" id="navbarResponsive">
        <ul class="navbar-nav ms-auto">
          <li class="nav-item mx-0 mx-lg-1"><router-link class="nav-link py-3 px-0 px-lg-3 rounded" to="/blog">Blog</router-link></li>
          <li v-if="!isLoggedIn" class="nav-item mx-0 mx-lg-1"><router-link class="nav-link py-3 px-0 px-lg-3 rounded" to="/registration">Sign Up</router-
link></li>
          <li v-if="!isLoggedIn" class="nav-item mx-0 mx-lg-1"><router-link class="nav-link py-3 px-0 px-lg-3 rounded" to="/login">Sign In</router-
link></li>
          <li v-if="isLoggedIn" class="nav-item mx-0 mx-lg-1"><router-link class="nav-link py-3 px-0 px-lg-3 rounded" to="/secure">Secure</router-
link></li>
          <li v-if="isLoggedIn" class="nav-item mx-0 mx-lg-1"><router-link @click="logout" class="nav-link py-3 px-0 px-lg-3 rounded"
to="/">Logout</router-link></li>
        </ul>
      </div>
    </div>
  </nav>
</template>
<script>
```

Depending on logged in status, display the correct links

On logout click invoke the logout method

Navigation.vue


```
<script>
import app from "../api/firebase";
import {getAuth, onAuthStateChanged, signOut} from "firebase/auth";

export default {
  name: "Navigation",
  data() {
    return {
      isLoggedIn : false
    }
  },
  created (){
    // Check if the user is logged in
    const auth = getAuth(app);
    onAuthStateChanged(auth, (user) => {
      if (user) {
        console.log(user);
        this.isLoggedIn = true;
      } else {
        this.isLoggedIn = false;
      }
    });
  },
  methods : {
    logout(){
      signOut(getAuth(app)).then(() => {
        // Send them back to the home page!
         this.$router.push("/");
      });
    }
  }
}
</script>
```

On page **created** check if the user is logged in, if so then set the isLoggedIn to **true**

Create a **logout** method to log users out and redirect them to the home page

Exercise 3

- Add a secure page and enable navigation guards to check the status.

- Edit Navigation.vue to check for logged in status and enable logout functionality and display the secure link if the user is logged in

Securing serverless function

- If the secure page requests a function to display some data, if we don't secure it, a malicious attacker could simply grab the URL and start making requests using Postman to get your data.
- Remember that the navigation guards are client-side protection but in reality it "should be" easy for an attacker to still access the "Secure" page.
- Thus it is best to make sure that all sensitive data delivered to that page is from a secure serverless function

HTTPS onRequest vs onCall

- Up until now we have been using the onRequest serverless function with Firebase

```
exports.getcomments =
functions.https.onRequest((request, response) => {
```

- These don't "out of the box" support authentication
- Another method **onCall** can instead be used which does

```
exports.securefunction = functions.https.onCall
```

onCall


```
exports.securefunction =
functions.https.onCall((data, context) => {

    // context.auth contains information about the
user, if they are logged in etc.
    if(typeof context.auth === undefined)
    {
        // request is made from user that is logged in
        return "User is logged in"
    }
    else
    {
        return "User is not logged in"
    }

});
```

Data contains data that is passed with the call from the client

Context contains the user information.

If **context.auth** is present then the request came from a logged in user

You can then return sensitive data to the page

functions>index.js

https://firebase.google.com/docs/functions/callable

Testing with Postman

- It's more challenging to test these functions with Postman as they require more headers to be set and authentication tokens to the passed for authenticated users

- The documentation is available here if you wish to do it https://firebase.google.com/docs/functions/callable-reference

onCall – Secure SFC


```
import app from '../api/firebase';
import { getFunctions, httpsCallable, connectFunctionsEmulator }
from "firebase/functions";
export default {
  name: "Secure",

  created(){
    // Call secure function and load some data
    const functions = getFunctions(app);
    if(window.location.hostname === 'localhost') // Checks if working
locally
      connectFunctionsEmulator(functions, "localhost", 5001);
    const secureFunction = httpsCallable(functions, 'securefunction');
    secureFunction()
        .then((result) => {
          // Read result of the Cloud Function.
          /** @type {any} */
          console.log(result);
        });
  }
}
```

Using the same httpsCallable import when we invoke our function from the client as a logged in user, the user data, i.e. userID, tokens etc. is all passed to the server.

If the user is logged out or unauthenticated then it no user info will be passed and the server can check for this.

Exercise 4

- Create a secure serverless function using the onCall method. Implement the function on slide 33.

- Invoke it from the client when logged in as well as logged out, to see the responses. In order to do it when logged out you will need to relax the navigation guard on routes.js to allow unauthenticated users to access the Secure.vue page

JWT

- Understanding tokens, JWT
- Understanding the Authentication flow
- Storing the access tokens in the cookie jar
  - Registration.vue
  - Login.vue
- Creating a utility function for access the cookie

Welcome to tokens!

- When you pay in to a concert or nightclub and you've shown your ID or ticket – you've authenticated

- You will often get a wristband, determining what areas you can access

In a similar way…

- Once a user authenticates successfully, they should be issued with a token.
- Every time they try to access an area of our site they present their token and which determines whether or not to give them access to the resource

Access tokens

- Once a client has authenticated with Firebase, Firebase issues an access token.

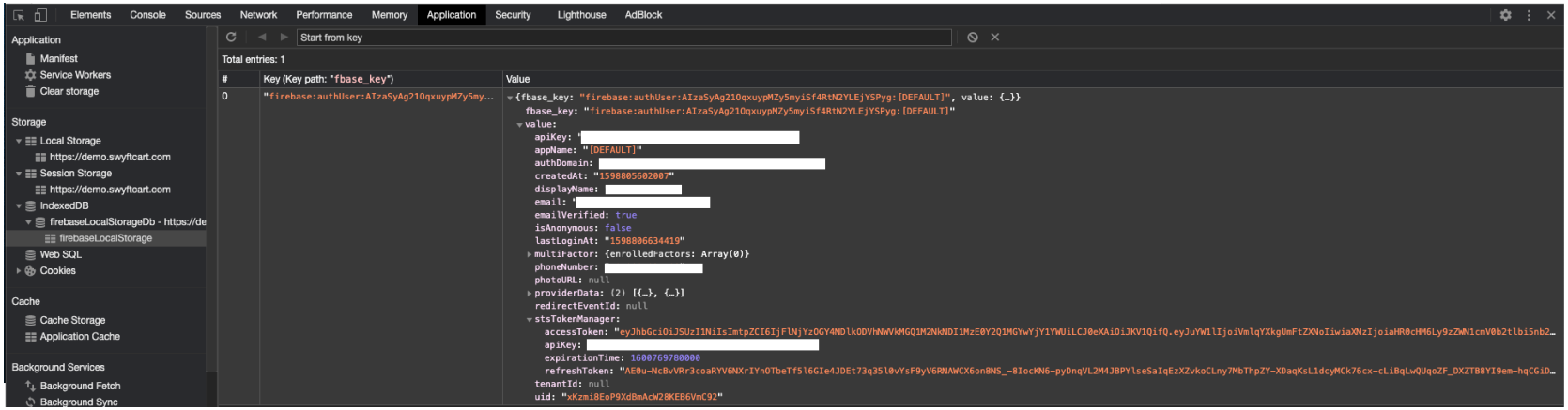
- This token contains the security credentials for the login session, identifies the user, possible groups etc.

Where does FB store this on the browser?

IndexedDB is a low-level API for client-side storage of significant amounts of structured data, including files/blobs. This API uses indexes to enable high-performance searches of this data. While Web Storage is useful for storing smaller amounts of data, it is less useful for storing larger amounts of structured data.

https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

https://i.stack.imgur.com/iroAK.png



Aside: Storing passwords

- Sony hack
  - Storing passwords in plain text
- Ashley Madison
- Cupid media
- Why is it such a bad idea?
- Firebase handles this for us with industry leading practices!

Overview: Cryptographic hashing

- Hashing is the transformation of a string of characters into a fixed length value or key that represents the original string.
- One way hashing function is used to protect passwords.

```
hash("enda") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1f
hash("enca") = 58756879c05c68dfac9866712fad6a93f8146f3366
hash("barr") = c0e81794384911616560b120fda8e90f383853542
```

A small change results in a completely different set of characters

Access token - JSON Web Tokens

- JWT consists of three parts:
  - Header, containing the type of the token and the hashing algorithm
  - Payload, containing the claims
  - Signature, bcrypt hashed token:

- This means that a token looks like the following

```
xxxxx.yyyyy.zzzzz
```

Bcrypt

- Bcrypt is a password hashing function based on the blowfish cipher designed by Niels Provos and David Mazières.

- It is the default password hashing function for a number of Linux distributions such as OpenBSD or SUSE

The generated JWT

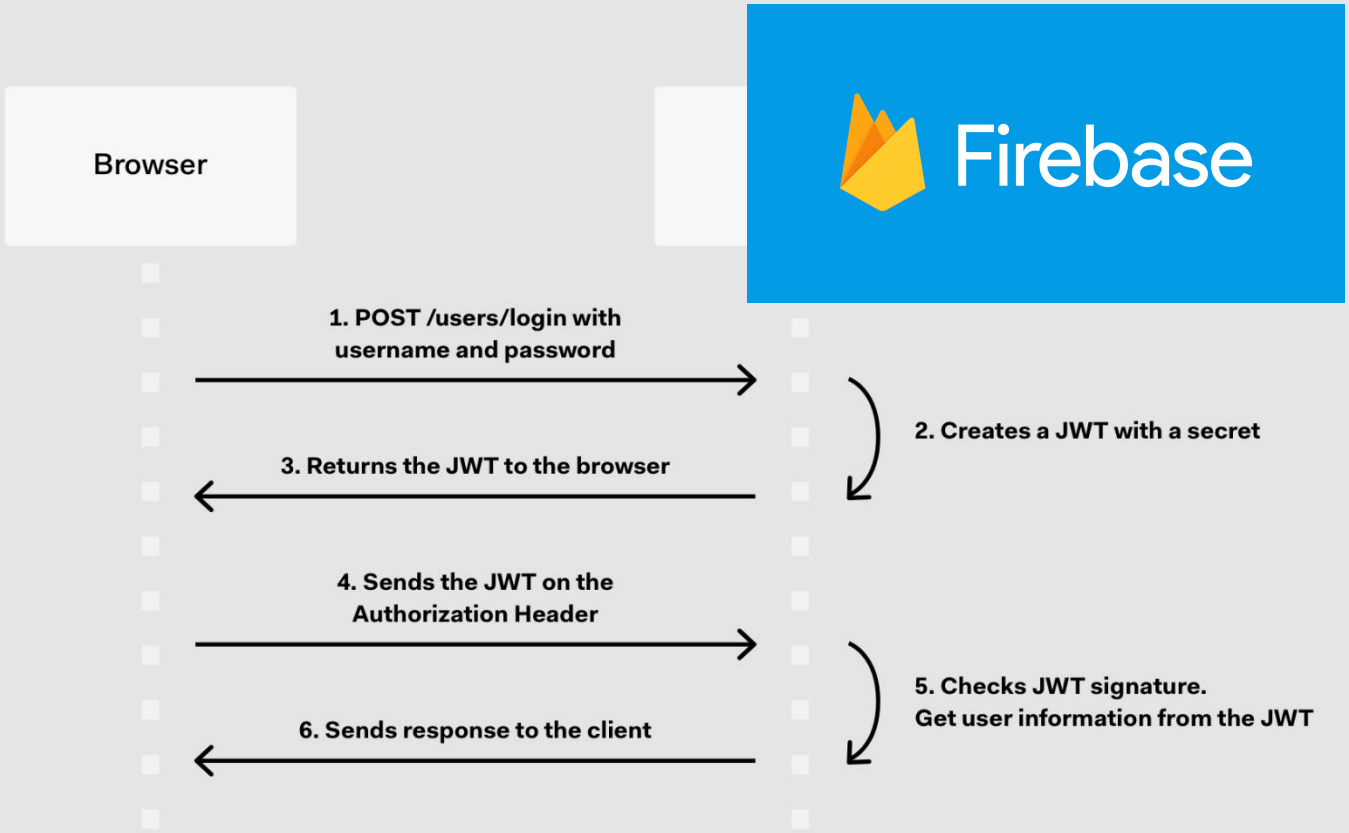

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX
25hbWUiOiJiYXJyZXR0ZW5kYUBlbWFpbC5jb20iLCJpYXQ
iOjE1MTAwNjgzMzEsImV4cCI6MTUxMDkzMjMzMSwiaXNzI
joiTXlzZWxmIn0.E63o90XX-
f6jDjxPmzfpjjeqZI1DDTh6tqksobLToq4
```

This is an example of a signed JWT.
Firebase will be generating these when the user logs in or registers and sending to the client in the user object.

Note the format of header, payload and signature

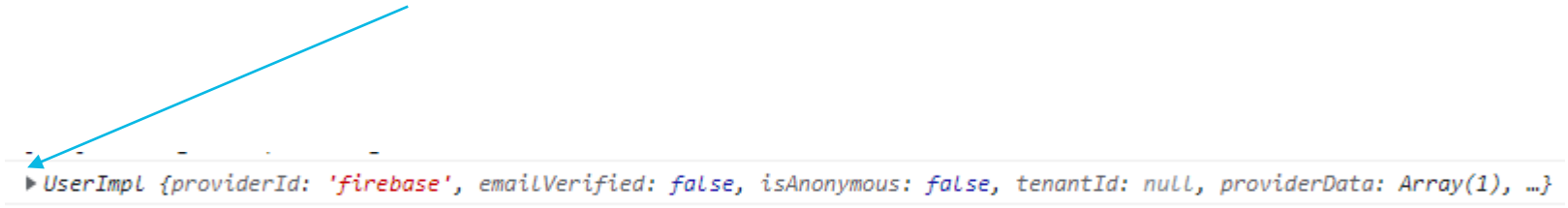
```
xxxxx.yyyyy.zzzzz
```

Authentication Flow

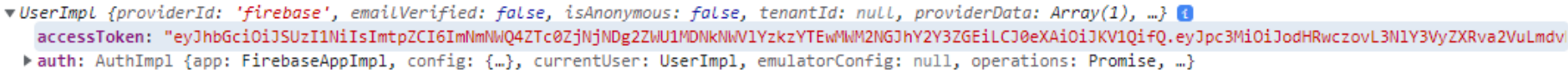


Check the browser console when logged in

- Expand the user object in the browser console



- The field at the top accessToken contains the access token!



USE CASE MODELING

Dr. Enda Barrett

Ollscoil na Gaillimhe
University of Galway

What is a use case?

"Represents an interaction between a user and a computer system, from the users perspective"

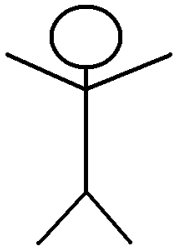
- It allows for the capturing of functional requirements
- Online shopping example:
  - User login
  - User registration
  - User adds items to a cart
  - User pays for item

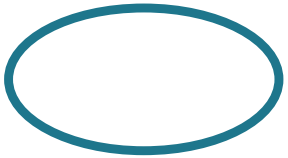
Main benefits of creating Use Cases

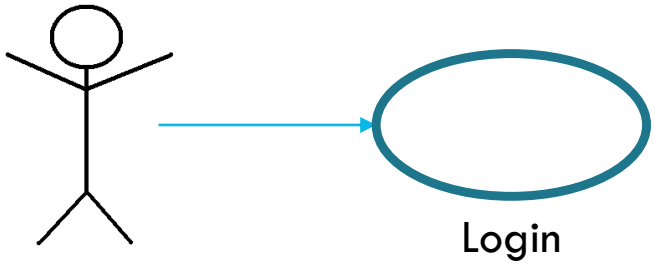
- Gives an insight/understanding of how the user will interact with the system and the requirements which must be included to ensure this happens
- Give the developers an insight as to problems that may arise
- Gives a good overview of the entire system at a high/abstract level
  - New team member wishing to understand various parts of the application, could start with the use case diagram.

Elements of a Use Case Model

- Actors
  - People or things that use a computer system
  - Can be external computer systems
- Use cases
  - A meaningful piece of functionality
- Relationships
  - Links between actors and use cases

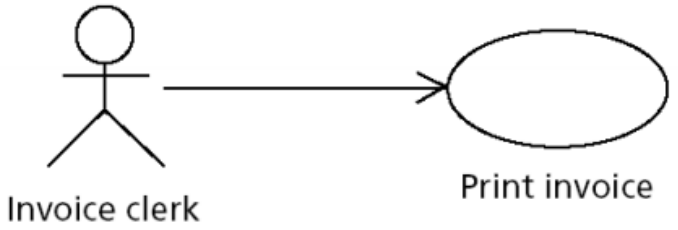




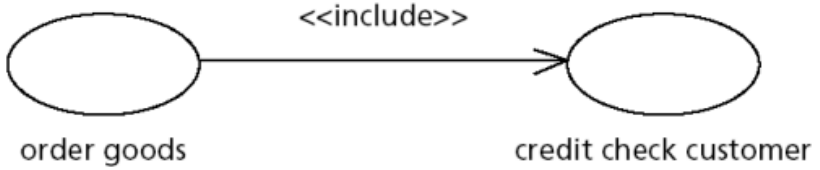


Use Case Model

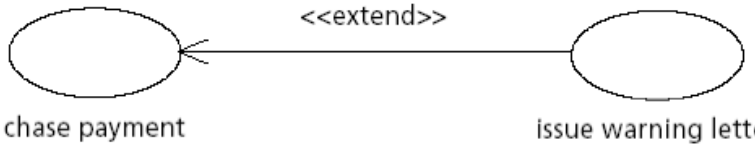
- Direction of initiation



- One use case "includes" another (always)



- One use case "extends" another (sometimes)



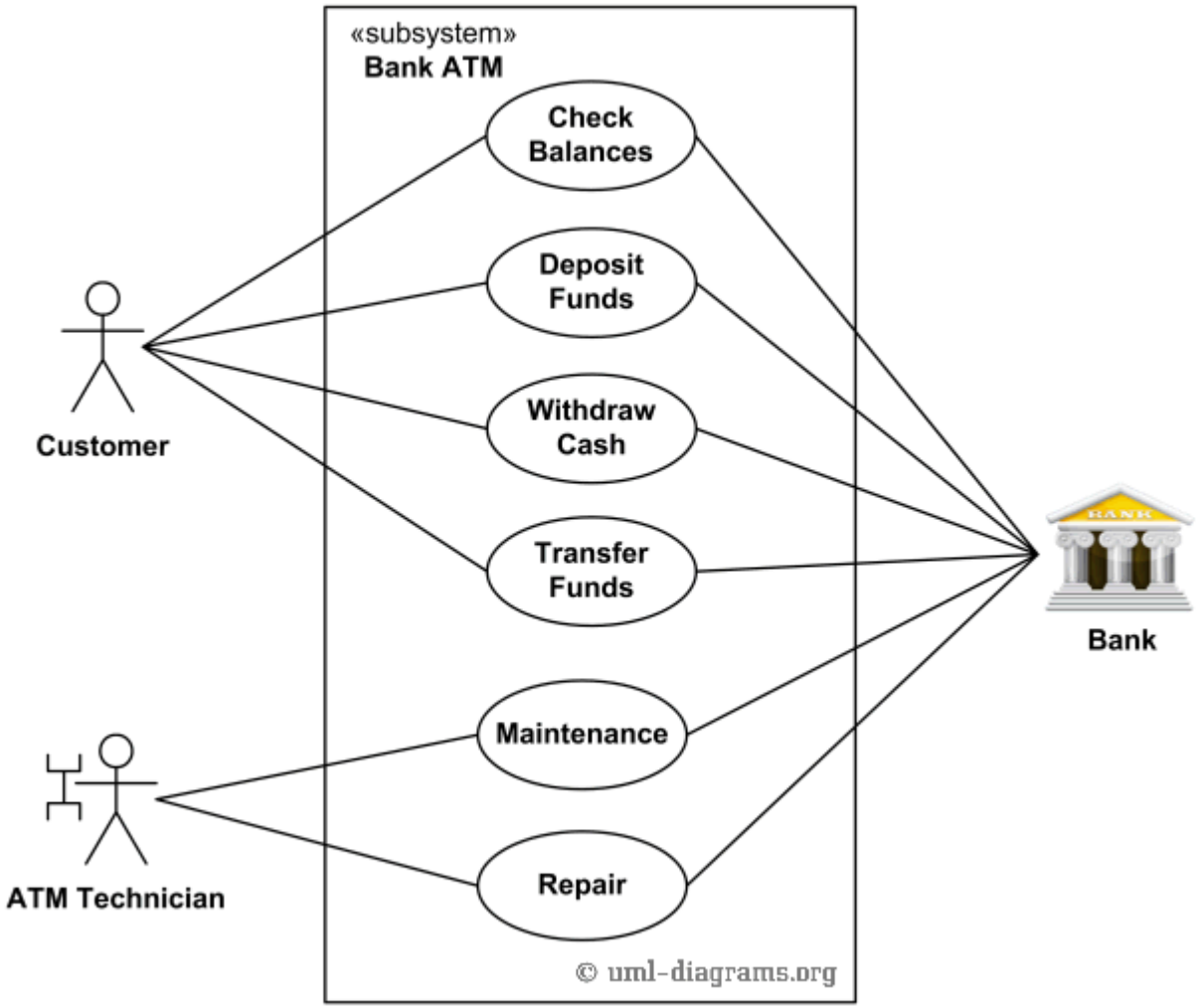
Use Case Model

- Important to remember that a Use Case Model comprises the Use Case Diagram **and** the Use Case Descriptions

Generating Use Cases

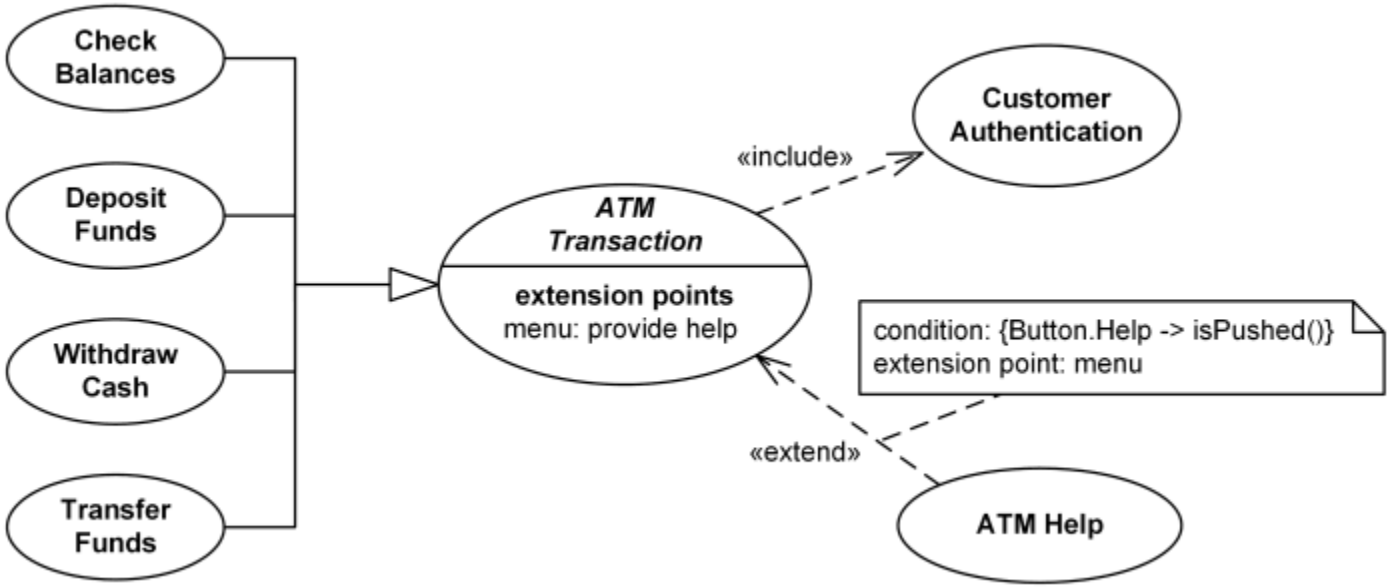
- From these steps in the business process
  - Primary
    - Sequence of steps that achieves the use case's goal
  - Alternative
    - An alternative flow is a step or sequence of steps that achieves the use case's goal following different steps than described in the main success scenario
  - Exception
    - An exception is anything that leads to NOT achieving the use case's goal.

Use cases to support an ATM system



Uml-diagrams.org

Customer Authentication use case



https://www.uml-diagrams.org/bank-atm-uml-use-case-diagram-example.html?context=uc-examples

Benefits of a Use Case driven process

- Use cases are concise, simple and understandable by a wide range of stakeholders
  - End users, developers and acquirers understand functional requirements of the system
- Use cases drive numerous activities in the process
  - Creation and validation of the design model
  - Definition of test cases and procedures of the test model
  - Planning of iterations
  - Creation of user documentation
  - System deployment
- Use cases help synchronise the content of different models

Template for a use case description

Use case number:	Use case name:
Goal:	
Brief description:	
Actors:	
Frequency of execution:	
Scalability:	
Criticality:	
Other non-functional requirements:	
Primary path:	
Use cases related to primary path:	
Alternatives:	
Use cases related to alternatives:	
Exceptions:	
Use cases related to exceptions:	
Notes:	

Authenticate User

Use Case no.: 1	**Use case name:** User Authentication
Goal: Ensure that a user is valid before other functions can be executed	
Brief description:	
Actors:	
Frequency of execution:	
Scalability:	
Criticality:	
…	

DATA OWNERSHIP, STATE MANAGEMENT

Dr. Enda Barrett

Ollscoil na Gaillimhe
University of Galway

Lecture overview

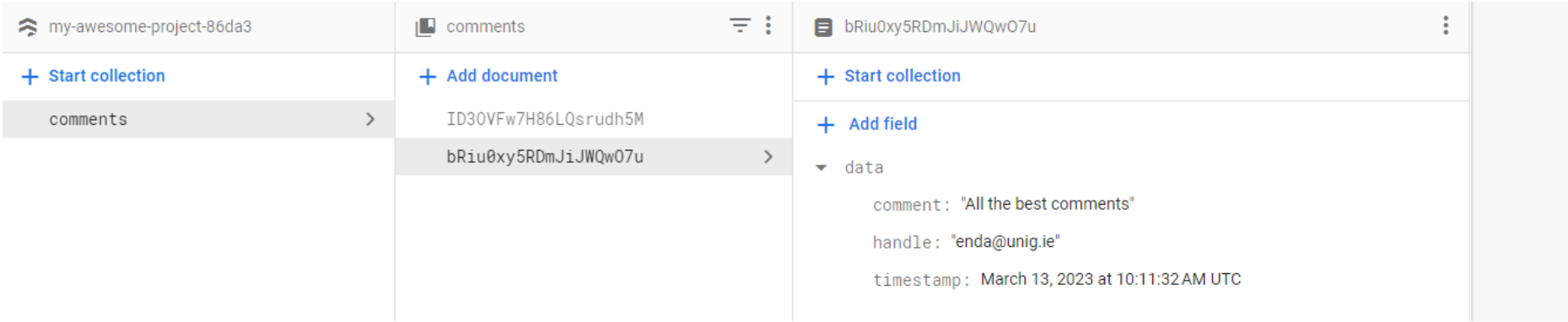
- Using Serverless functions to assign ownership to documents
- Creating and Deleting documents owned by a particular user
- State management
  - Technical examples and discussion

Logged in users posting comments

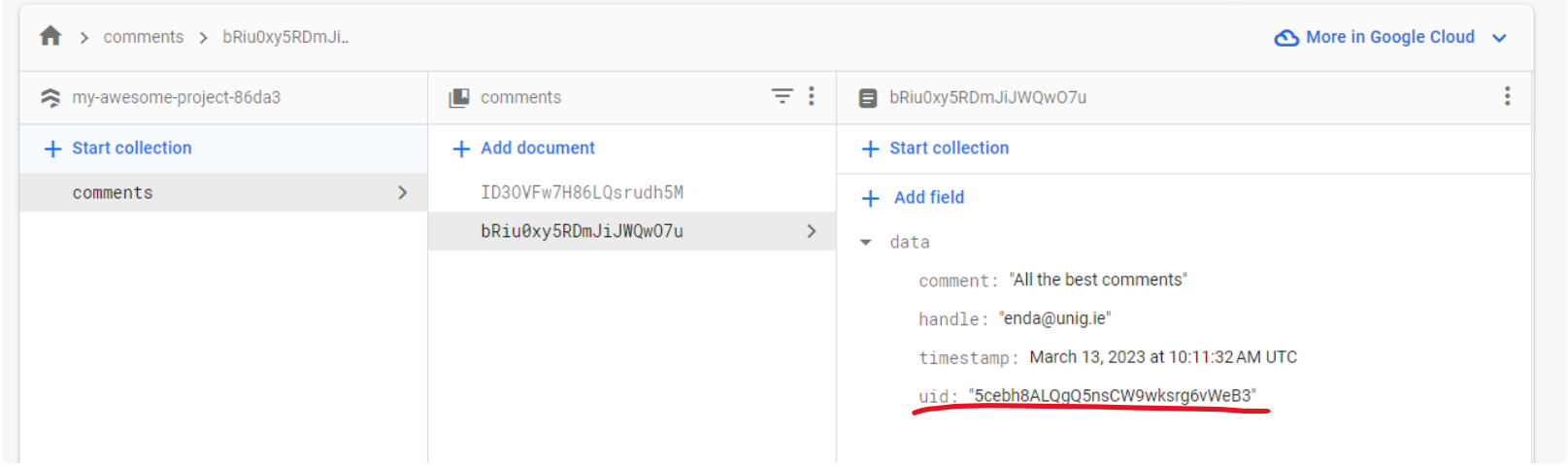
- Up until now posting comments on our application was anonymous, any user could log in and post a comment simply be specifying their handle, typing their comment and submitting it to the server.
- Since we now have authentication enabled, we can automatically the handle with the users email address and assign using their UID we can assign ownership of that document to a particular user

Comment data structure

Current data structure/schema



Adding the UID to the document allows us to denote ownership



Document ownership

- Goal: Ensure that the logged in user who creates a comment has their UID associated with that document, that only they can edit and delete their comment.
- There are two approaches to doing this
  - First approach is to convert our CRUD (*postcomment etc.*) functions to onCall serverless functions and check the user auth each time.
  - Let us examine this approach first

Posting user comments

- Switching postcomment to oncall from onrequest allows us to check the c*ontext* object for a logged in user.

- If found then create the document but make sure to include the UID in the document also.

- Let's alter the client code on Blog.vue to check for loggedIn status

Add a user property


```
export default {
  data() {
    return {
      handle: '',
      comment: '',
      commentsArray:[],
      editing:false,
      tempValue:null,
      user:null
    }
  },
```

Add a user property to the page which will hold the user object of logged in users. At anytime we can check if a user is logged in just by checking if this value is null.

pages/Blog.vue

Update the created method


```
created(){
  // Check for logged in user
  const auth = getAuth(app);
  onAuthStateChanged(auth, (user) => {
    this.user = user; // set the user object to the user prop
    if (user) {
      console.log("User", user);
      // User is signed in
    } else {
      console.log("No user found")
      // User is not signed in
}
  });
  this.getComments();
  //window.setInterval(this.getComments, 1000);
},
```

pages/Blog.vue

postusercomment


```
exports.postusercomment = functions.https.onCall((data, context) => {

    // context.auth contains information about the user, if they are logged in
etc.
    const currentTime = admin.firestore.Timestamp.now();
    data.timestamp = currentTime;

    if(typeof context.auth === 'undefined')
    {
        // request is made from an anonymous user
        return admin.firestore().collection('comments').add({data:
data}).then(() => {
            return "Data saved in Firestore"
        });
    }
    else
    {
        data.uid = context.auth.uid;
        return admin.firestore().collection('comments').add({data:
data}).then(() => {
            return "Data saved in Firestore"
        });
    }

});
```

Irrespective of whether the request is made from a logged in user or an anonymous user a document is written to the *comments* collection

The uid is added to the documents for requests originating from logged in users

functions/index.js

postComment

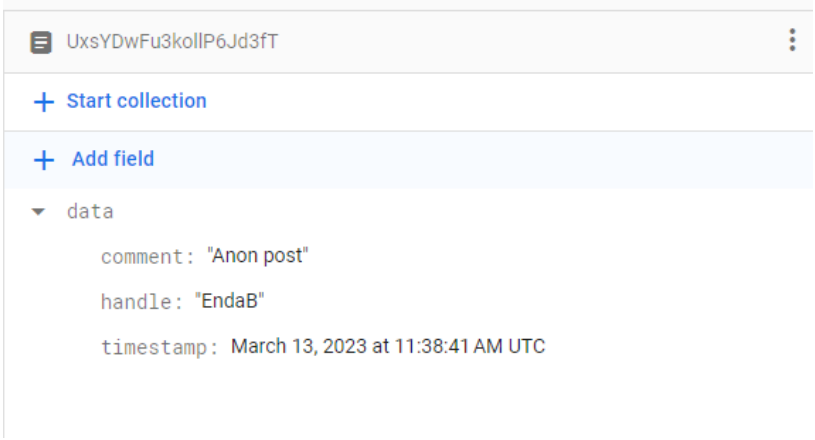

```
methods : {
  postComment() {
    let loader = this.$loading.show({    // Optional parameters
      loader: 'dots',
      container: this.$refs.container,
      canCancel: false
    });
    const functions = getFunctions(app);
    // Uncomment this code if your local emulators are running and you
wish to test locally
    //if(window.location.hostname === 'localhost') // Checks if working
locally
      //connectFunctionsEmulator(functions, "localhost", 5001);
    const postComment = httpsCallable(functions, 'postusercomment');
    postComment({"handle": this.handle, "comment":
this.comment}).then((result) => {
      // Read result of the Cloud Function.
      // /** @type {any} */
      loader.hide();
      this.getComments();
    });
  },
```

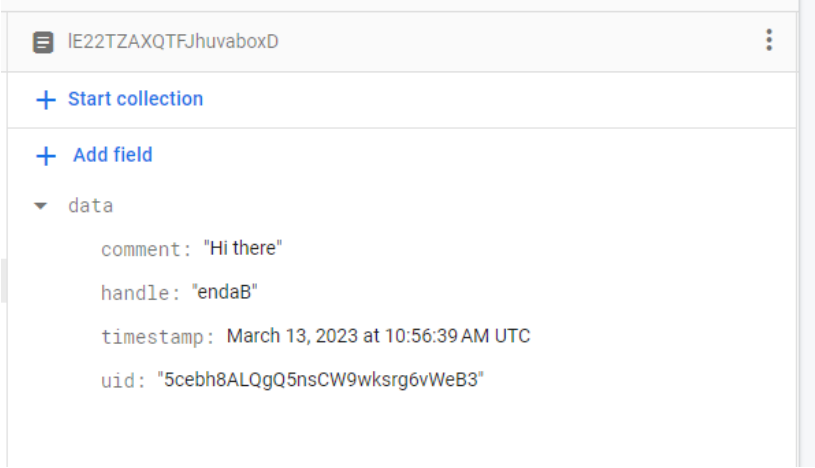
postComment is modified to invoke the new *postusercomment* function different function

pages/Blog.vue

Check the Firestore DB

- Test the application by submitting a comment as a logged in user and an anonymous user and see the different documents in the database





Auto populate handle


```
created(){
  // Check for logged in user
  const auth = getAuth(app);
  onAuthStateChanged(auth, (user) => {
    this.user = user;
    if (user) {
      console.log("User", user);
      this.handle = user.email;
      // User is signed in
    } else {
      console.log("No user found")
      // User is not signed in
    }
  });
  this.getComments();
  //window.setInterval(this.getComments,
1000);
},
```

If the user has an account and is logged in, then we can set the handle to their email address when the page loads.

However the user can still edit this on the client if they wish!

pages/Blog.vue

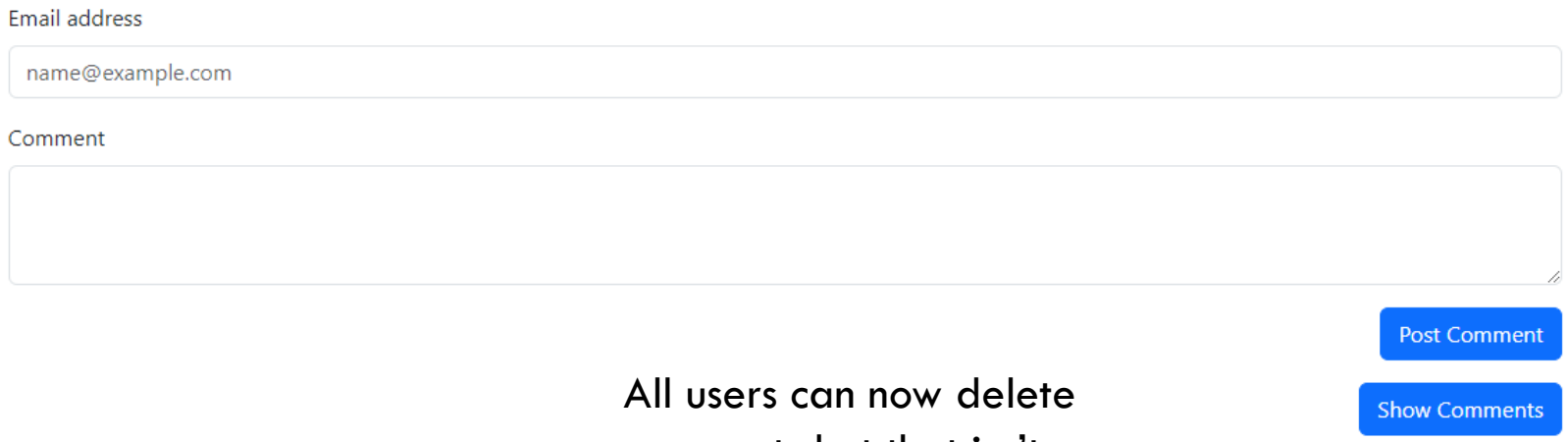
Exercise 1

- Modify postcomment to include the uid in each document created for each new comment. If the user is not logged in then no uid field needs to be set.

- Note that the uid is set server side, could we have set this on the client and sent it along with the data request

```
 const postComment = httpsCallable(functions,
'postusercomment');
    postComment({"handle": this.handle, “uid": user.uid,
"comment": this.comment}).then((result) => {
      // Read result of the Cloud Function.
      // /** @type {any} */
      loader.hide();
      this.getComments();
    });
```

Deleting comments



All users can now delete comments but that isn't a really great idea. Only logged in users should be able to delete their own comments.

The same could also be said for editing etc.

deleteusercomment


```
exports.deleteusercomment = functions.https.onCall((data, context) => {

    if(typeof context.auth === 'undefined')
    {
       // request is made from an anonymous user
        throw new functions.https.HttpsError('permission-denied', 'Anonymous users cannot delete
comments');
    }
    else
    {
        return admin.firestore().collection('comments').doc(data.id).get().then((doc) => {
            if (!doc.exists) {
                // 1. Check if the document exists, throw error if not
                throw new functions.https.HttpsError('not-found', 'No comment found matching the
id');
            } else if (doc.data().data.uid != context.auth.uid) {

                // 2. Check if the user owns the document, otherwise throw error
                throw new functions.https.HttpsError('permission-denied', 'You do not have sufficient
permissions to delete this comment');
            } else {
                // 3. If the user created the document then delete it
                return doc.ref.delete().then(() => {
                    return 'Document successfully deleted'
                });
            }
        });
    }
});
```

Check the user id to make sure the request is coming from a logged in user.

Only allow those that created a particular comment to delete it.

functions/index.js

Delete comment on Blog.vue


```
deleteComment(id){
  const functions = getFunctions(app);
  // Uncomment this section if your local emulators are running
and you wish to test locally
  //if(window.location.hostname === 'localhost') // Checks if
working locally
  //connectFunctionsEmulator(functions, "localhost", 5001);
  const deleteComment = httpsCallable(functions,
'deleteusercomment');
  deleteComment({id:id}).then((result) => {
    console.log(result.data);
    if(result.data == "Document successfully deleted")
      this.getComments();
  }); // To refresh the client
},
```

Change the function name to point the call to our new function.

Pass the id to the serverless function

pages/Blog.vue

Exercise 2: Deleting comments

- Implement the delete comments functionality, where a user can only delete those comments that are owned by them.
- Disable the deleted buttons for all anon users.
- Display the deleted button beside only those comments owned by a particular user.

State management

- Sometimes it is very useful to have a mechanism through which multiple SFCs can share data. Often this is known as sharing “state”.
- It’s also very useful to store information client side. For example, once you have made a request to the server for data (such as comments) wouldn’t it be nice to be able to keep that data “somewhere” on the client
- This is sometimes called a “store pattern”
- The most recent state mgmt. solution they support is called Pinia
  - https://pinia.vuejs.org/

Simple store pattern

- Sharing information between views
  - Because everyone is building different apps, it is difficult to define a use case for this to suit all apps
  - Some ideas could be to share a handle between pages, share data, UI preferences etc.
  - In the past, I've used them exclusively as a sync'd data layer which keeps in sync with the database. Mutations are automatically propagated back to DB and method calls will return data from the store rather than making DB calls.

https://vuejs.org/guide/scaling-up/state-management.html

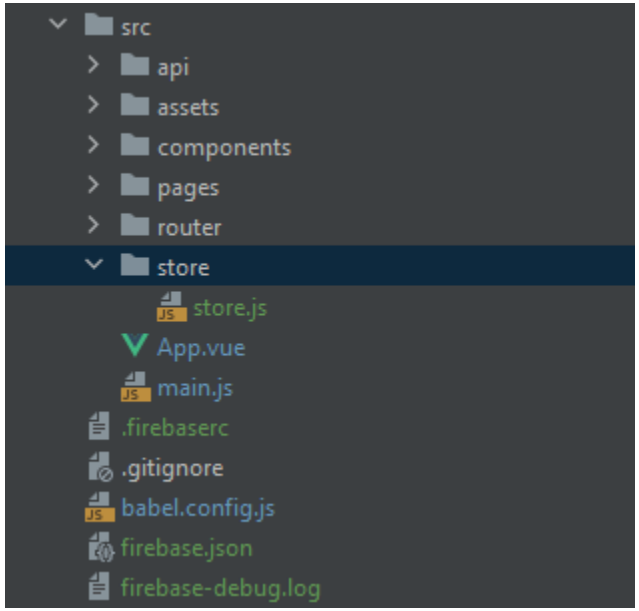
Generic counter example

- Imagine you wished to increment a shared counter from two UIs. I'm not creative enough to understand why you would do this, but it's the default example often given on the doc pages for frameworks like React, Angular and VueJS.
- The store pattern is an ideal solution for this.

- It relies on a specific API provided by VueJS known as the "Reactivity API" which creates a reactive object that can be shared amongst multiple SFCs

Create new directory “store”

- Under the src directory, create a new directory called store



https://vuejs.org/guide/scaling-up/state-management.html#what-is-state-management

Import the reactive API

- A store can be created by passing an object as an argument to the constructor of the reactive API

- In this case we have simple count

```
import { reactive } from 'vue'

export const store = reactive({
  count: 0,
  increment() {
    this.count++
  }
})
```

src>store>store.js

Method to increment the counter

https://vuejs.org/guide/scaling-up/state-management.html#what-is-state-management

Import the store to our first SFC

- Let's add a click counter onto the bottom of our Blog.vue page.
- Start by importing the store

```
import {store} from '../store/store';

export default {
 name: "Blog",
 data(){
  return {
   handle:'',
   comment:'',
   comments:[],
   tempValue:"",
   editing:false,
   store
  }
 },
```

pages>Blog.vue

Add a reference to it

Utilise the counter in Blog.vue

Display the store count

```
<div>
  Counter {{store.count}}
  <button type="button" @click="store.increment()" class="btn btn-primary">Click Counter {{store.count}}</button>
</div>
```

Method to increment the counter

pages>Blog.vue

Create a new SFC – Blog2.vue


```
<template>
  <div>
    Counter {{store.count}}
    <button type="button"
@click="store.increment()" class="btn btn-
primary">Click Counter</button>
  </div>
</template>

<script>
import {store} from '../store/store';

export default {
  name: "Blog2",
  data(){
    return {
      store
    }
  }
}
</script>

<style scoped>

</style>
```

In order to test the reactivity, one can create a separate SFC called Blog2.vue

Add the component to the page


```
<div>
  <Blog2 />
</div>
```

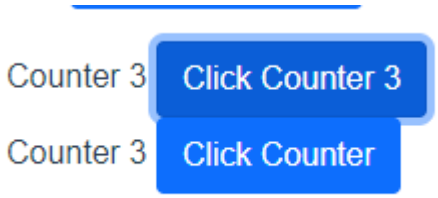
Add to the template

```
<script>
import Blog2 from "./Blog2";
```

Import component

```
,
components: {
  Blog2
 },
```

Reference it in order to render it on the Blog page



pages>Blog.vue

Things you could use the store for

- User details. Once a user has signed in you could pop the *user object* into the store and then access it on any page that needs it.

- Comment data. If you need to use comments on multiple pages you can store the comments in the store and make local calls to retrieve them.

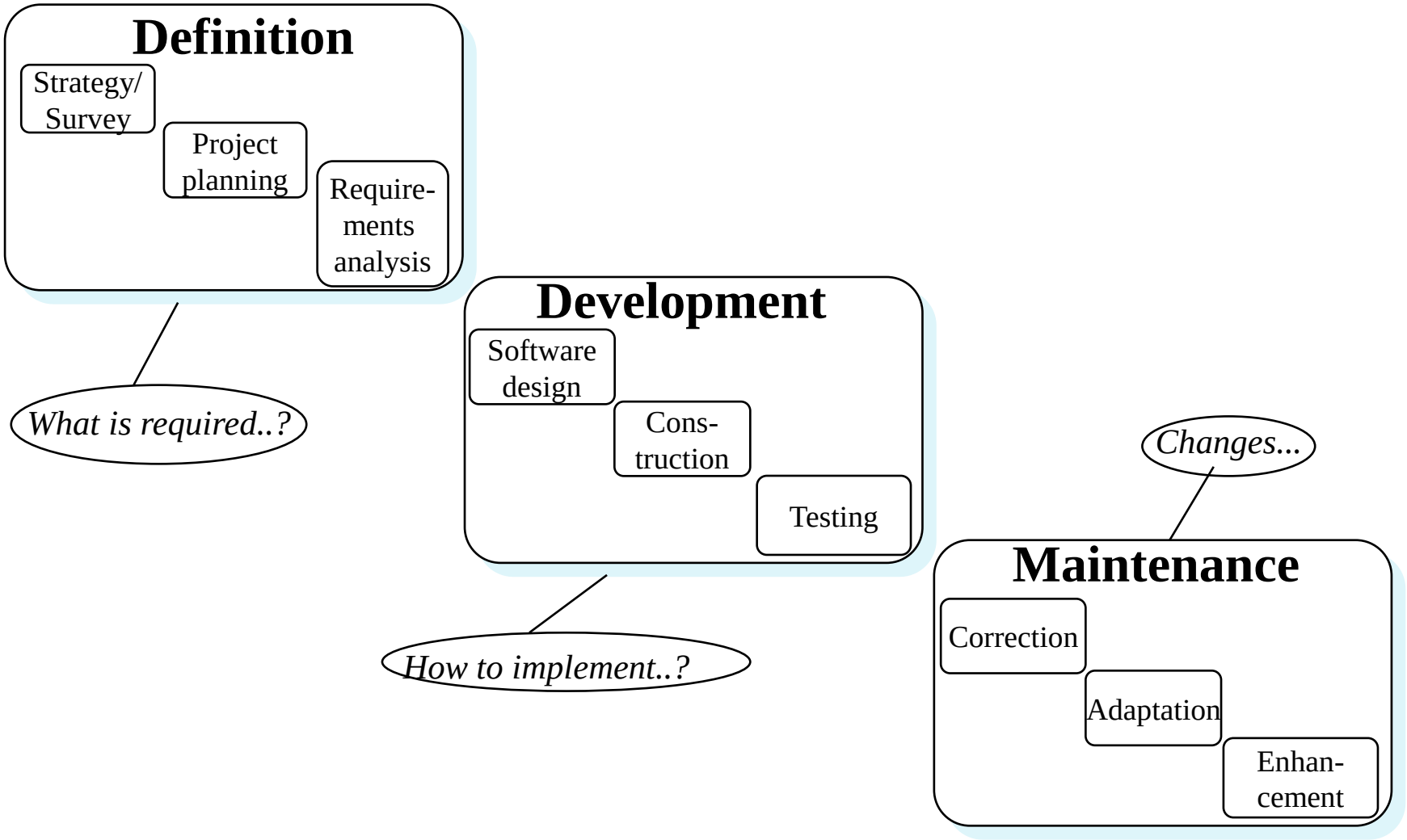
SOFTWARE DEVELOPMENT PARADIGMS

Dr. Enda Barrett

Software Development Paradigms

- Paradigms of software development
  - Software life cycle
- A few flavours of the waterfall model
  - Traditional Waterfall
  - Waterfall with feedback
  - Waterfall with feedback and overlaps
- Prototyping

SDLC


Definition
Strategy/ Survey
Project planning
Require-ments analysis
What is required..?
Development
Software design
Cons-truction
Testing
How to implement..?
Changes...
Maintenance
Correction
Adaptation
Enhan-cement

Software Development Lifecycle

- The software lifecycle is an abstract representation of a software process. It defines the steps, methods, tools, activities and deliverables of a software development project. The following **lifecycle phases** are considered:
  - 1. requirements analysis
  - 2. system design
  - 3. implementation
  - 4. integration and deployment
  - 5. operation and maintenance

1. Requirements Analysis

- User requirements are statements in natural language plus diagrams of what services the system is expected to provide and the constraints under which it must operate (Sommerville 2001)
- Activity of determining and specifying requirements
  - Interview users, try to make communication clear
  - Questionnaires to users
  - Observations of users performing their tasks
  - Study existing system documents
  - Study similar software systems to learn about domain knowledge
  - Prototypes to confirm requirements
- Output is a **requirements document**

2. System Design

- A software design is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between the system components and the algorithms used (Sommerville 2001)
  - Algorithms are not always concretely defined in this phase as it is necessary to give some freedom to the developers.
  - Design begins where analysis ends.
  - Theoretically : Analysis is modelling unconstrained by any hardware/software considerations
  - Design is modelling that takes into consideration the platform upon which it is to be deployed.

- The output of this phase should be a **design document**

3. Implementation

- Implementation is mostly, programming, testing
  - A programmer is a "*component engineer*"
  - The programmer will attempt to re-use code where ever they can. However, it can often require quite a bit of modification to suit the specific needs.
  - Expand upon the design, i.e. algorithms will only be partially specified, engineers will have to thrash out the specifics of the design.
  - Turn this design into code
  - Debugging
  - Testing
- Output is **code**

4. Integration and Deployment

- **Integration** assembles the application from the set of components previously implemented and tested.
- ***Deployment*** is the handing over of the system to the customer for production use.
- Integration can be sometimes difficult to disassociate from the implementation phase nowadays, especially with continuous integration tooling so readily used. But still a stage in it own right.
- Software is deployed in releases i.e. version 1.0, 1.1 …
- Deployment releases
  - Alpha
  - Beta

5. Operation and Maintenance

- The new software product is used in day-to-day operations while the previous system is phased out.
  - Often systems will be run in parallel during the phase out
- **Maintenance** (Maciaszek)
  - *Corrective* – fixing defects and errors discovered in operation (*patches*)
  - *Adaptive* – modifying the software in response to changes in the computing and business environment (*new release 7.6 to 7.7*)
  - *Perfective* – evolving the product by adding new features (depending on the features but maybe a whole new release *i.e. go from 7.6 to 8.0*)
- Finally the system only becomes a ***legacy system*** and are only retired when it is not technically possible to support them anymore
  - Enterprises will flog them till their death

Waterfall model

- When these phases are all linearly placed, we get the traditional waterfall model
- Came out in the 70's as a solution monolithic type business applications (Cobol), fixed number of subroutines
- The completion of each phase results in some deliverable.
- Today it is used much less frequently as trying to maintain strictly within the phases is difficult
- May take a long time to see the final product

Traditional Waterfall model

User

Requirements Analysis

System Design

Implementation

Integration and Deployment

Product release

Operation and Maintenance

Waterfall model

- Advantages
  - Documentation is produced at each phase
  - It fits in with other engineering process models
  - Easier to project manage
  - Sign-off phase clarifies legal position

- Disadvantages
  - Inflexible partitioning of the project into distinct phases
  - Difficult to adjust to changes in requirements
  - Delivered project may need re-work
  - Documentation can give false sense of the progress

Waterfall model

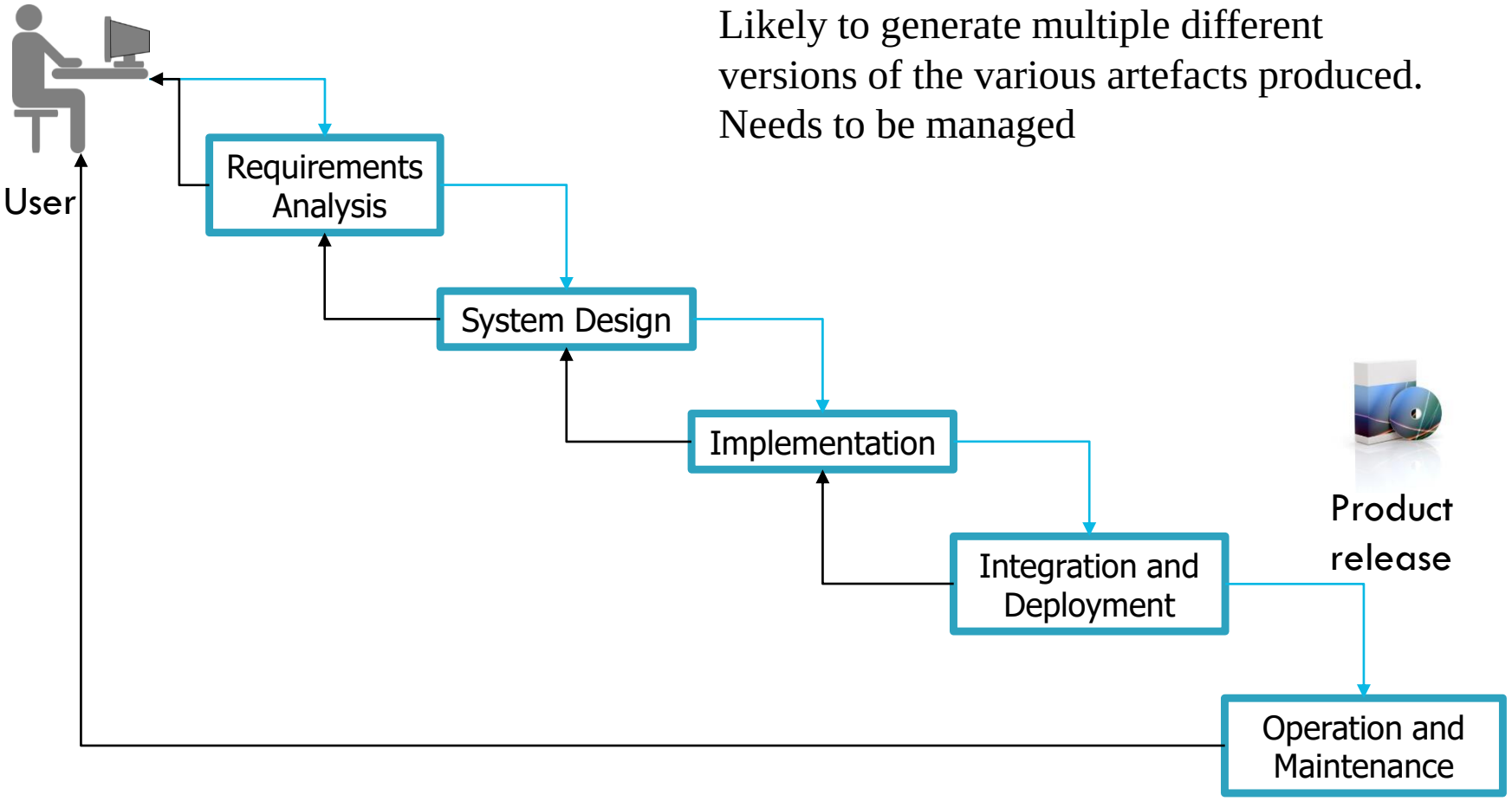
- An organisation may elect to use a particular lifecycle model to develop its software.
- However specifics of the lifecycle model i.e. how the work is done varies from org to org, from project to project.
- Remember a software product is not manufactured, it's "developed"
  - Every time you repeat the process you could easily get different results
  - This is why there are still so many engineering jobs and will continue to be for the foreseeable future ☺

Waterfall model

- Disadvantages
  - **Inflexible partitioning of the project into distinct phases**
  - **Difficult to adjust to changes in requirements**
  - Delivered project may need re-work
  - Documentation can give false sense of the progress

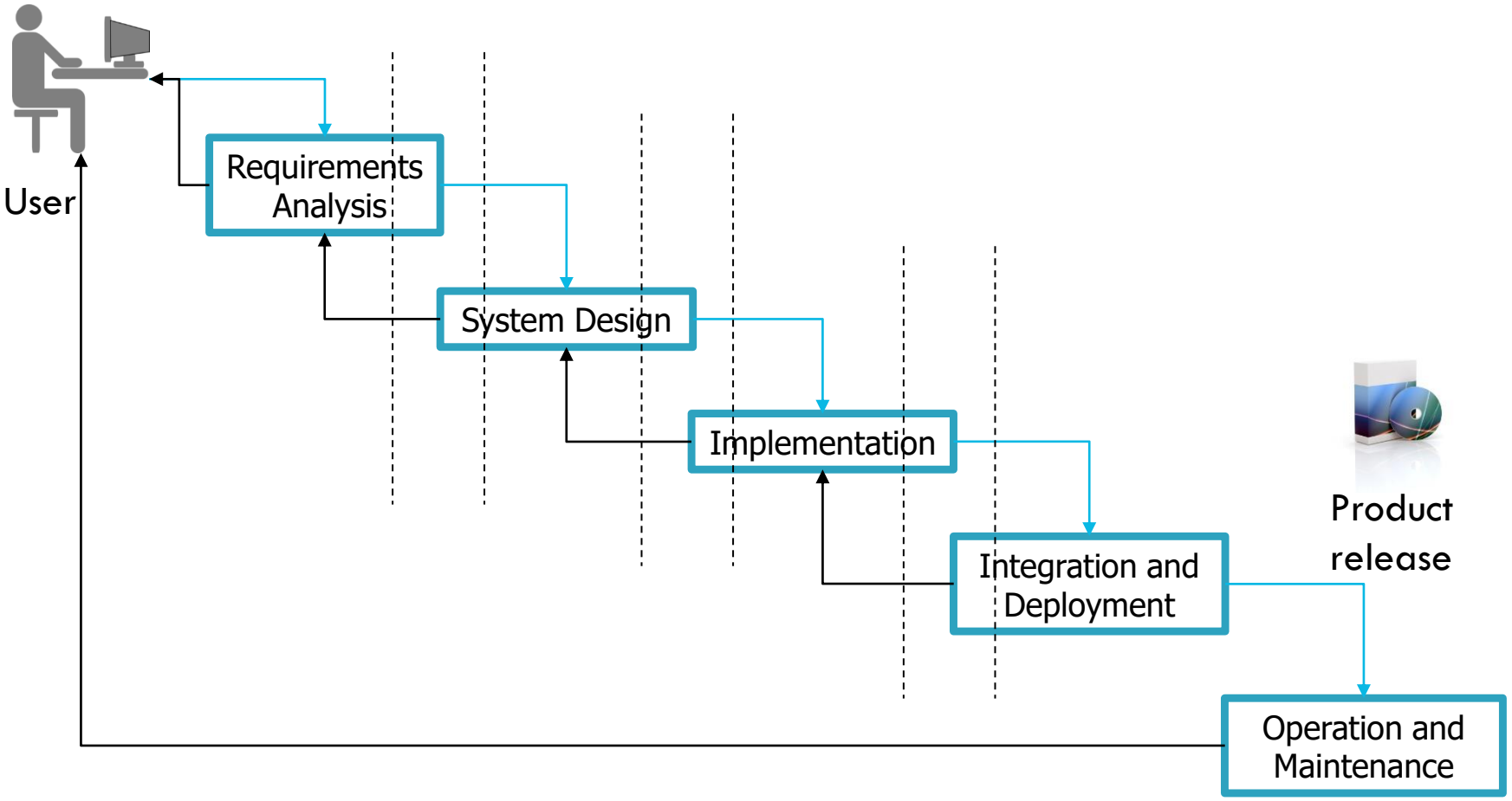
- Simplest way of addressing these two disadvantages is to introduce feedback paths to the development process

Waterfall model with feedback



Likely to generate multiple different versions of the various artefacts produced. Needs to be managed

Waterfall model with feedback and overlaps



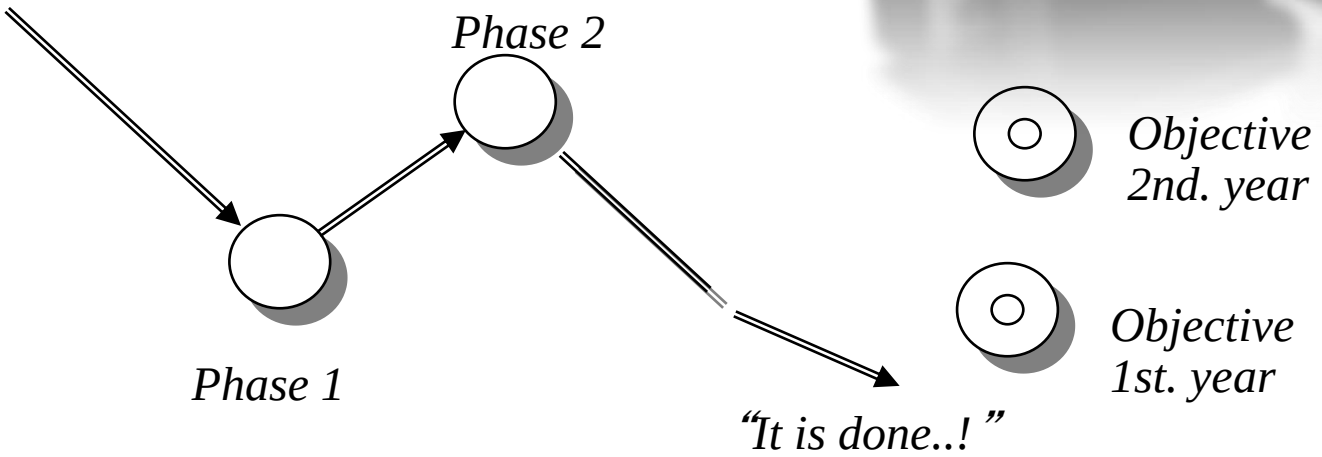
Deliverables from the SDLC

Phase	Deliverables
Requirements Analysis	Requirements specification Functional specification Acceptance test specification
System Design	Software architecture specification System test specification Design specification Sub-system test specification Unit test specification
Implementation & Testing	Program code Unit test report Sub-system test report System test report Acceptance test report Completed system
Integration & Deployment	Installed system
Operation & Maintenance	Change requests Change request report

Limitations of the SDLC

"The water isn't flowing....":

- Rarely delivers what is wanted
- Difficult to execute -- e.g. freezing specs.
- It takes too long
- Too much documentation

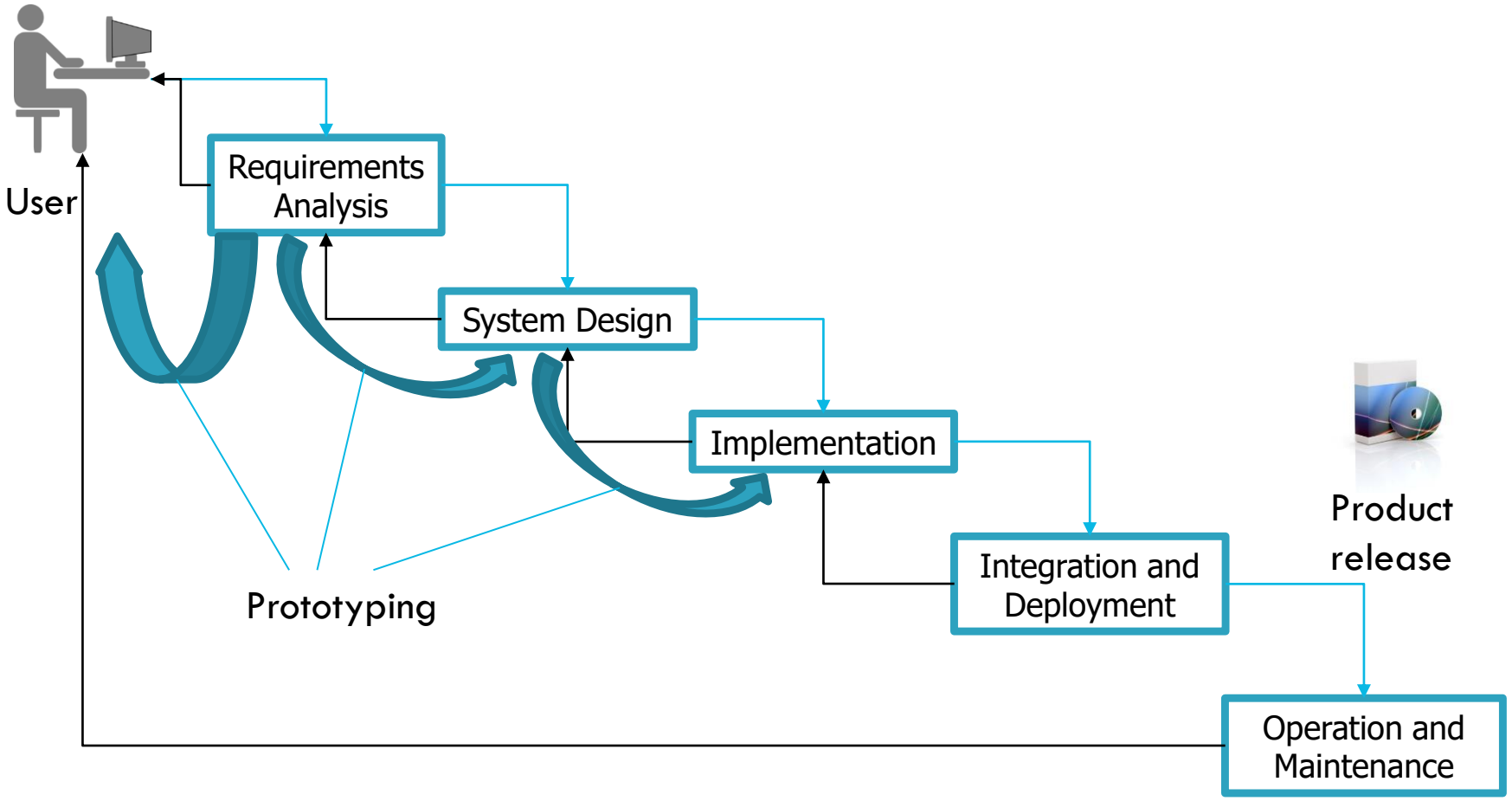


Prototyping

An alternative approach to requirements definition is to capture an initial set of needs and to implement quickly those needs with the stated intent of iteratively expanding and refining them as mutual user/developer understanding of the system grows.

Definition of the system grows through gradual and evolutionary discovery as opposed to omniscient foresight.

Waterfall model with prototyping



EXAM PREP

Dr. Enda Barrett

Exam Information

- Two hours in duration
- Answer any 3 questions out of 4, with all questions of equal marks
- Question 1: Software Engineering topics include testing. Weeks 1, 2, 3, 4, 19, 23.
- Question 2: HTML, CSS, Bootstrap. Weeks 6, 7.
- Question 3: Client-side JS and VueJS. Weeks 8,9,13,14,15,16
- Question 4: Server-side JS (NodeJS). Weeks 10, 11, 18, 20.